Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Estudo Sobre a Utilização do Banco de Dados NoSQL Cassandra em Dados Biológicos

Rodrigo Cardoso Aniceto
Renê Freire Xavier

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2014

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenadora: Prof.ª Dr.ª Maristela Terto de Holanda

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Edna Dias Canedo — FGA/UnB
Prof.ª Dr.ª Maria Emilia M. T. Walter — CIC/UnB

CIP — Catalogação Internacional na Publicação

Aniceto, Rodrigo Cardoso.

Um Estudo Sobre a Utilização do Banco de Dados NoSQL Cassandra em Dados Biológicos / Rodrigo Cardoso Aniceto, Renê Freire Xavier. Brasília : UnB, 2014.
117 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2014.

1. SGBDs, 2. Nuvem Computacional, 3. NoSQL, 4. Cassandra,
5. Dados Biológicos, 6. DataStax

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um Estudo Sobre a Utilização do Banco de Dados NoSQL Cassandra em Dados Biológicos

Rodrigo Cardoso Aniceto
Renê Freire Xavier

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof.ª Dr.ª Edna Dias Canedo
FGA/UnB

Prof.ª Dr.ª Maria Emilia M. T. Walter
CIC/UnB

Prof.ª Dr.ª Maristela Terto de Holanda
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 27 de fevereiro de 2014

Dedicatória

Dedico a minha família, amigos e todos aqueles que buscam o avanço da tecnologia para uma sociedade melhor.

"Success is not final, failure is not fatal: it is the courage to continue that counts." Winston Churchill

Agradecimentos

Agradeço a Deus, minha família e amigos que com muita paciência souberam lidar com a dedicação para este trabalho. Agradeço também a Doutora Maristela Terto de Holanda que nos apoiou, acompanhou e esteve sempre presente para nos instruir. Por fim agradeço ao meu companheiro de trabalho, não só pela ajuda aqui, mas também pelos ótimos momentos de descontração em meio a tanto trabalho.

Eu também.

Resumo

Por meio do avanço das tecnologias voltadas a escalabilidade de um banco de dados, o uso de bancos voltados a performance que administram uma massiva quantidade de dados tornou-se mais acessível. Esses bancos de dados são conhecidos como bancos NoSQL, e são um novo paradigma computacional. De forma teórica, este trabalho apresenta vantagens e desvantagens desses bancos e aprofunda-se em caraterísticas do Cassandra.

Em termos práticos, tendo em vista que os dados gerados pela bioinformática são de grande volume e necessitam de um armazenamento de alta performance, esse trabalho propõe o armazenamento destes dados no banco NoSQL Cassandra. Para isto foram realizados dois estudos de caso, um com duas e outro com quatro máquinas trabalhando em um mesmo banco do Cassandra. Estes estudos mostraram um avanço ao escalar os recursos físicos do banco, tornando a inserção e a consulta mais eficientes, sendo este uma alternativa para dar suporte aos projetos na área de bioinformática.

Palavras-chave: SGBDs, Nuvem Computacional, NoSQL, Cassandra, Dados Biológicos, DataStax

Abstract

Through the advancement of technologies focused on the scalability of a database, the use of databases oriented to performance that manage massive amounts of data has become more accessible. These databases are known as NoSQL, and are a new computing paradigm. This work presents advantages and disadvantages of these NoSQL databases and delves into features of Cassandra.

In the practical part, given that the data generated by bioinformatics are a large volume and need a high-performance storage, this work proposes the storage of such data in a NoSQL database, Cassandra. It were made two case studies, one with two machines working on the same database of Cassandra and the other with four machines working similarly the first one. These studies showed an improvement in scaling the physical resources of the database, making more efficient operations on the database, working as an alternative to support researches in the field of bioinformatics.

Keywords: SGBDs, Cloud Computing, NoSQL, Cassandra, Biological Data, DataStax

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

De modo breve, os bancos de dados são definidos como um conjunto de dados relacionados entre si armazenados segundo uma determinada estrutura de forma que possam ser recuperados quando necessário. Eles costumam ter um sistema responsável por esse armazenamento e para o controle dos dados [31].

Entre os diferentes paradigmas de Bancos de Dados, existe um que se destaca como uma nova opção que está em crescimento, trata-se da abordagem não relacional voltada para o problema e não para a padronização. Esse paradigma visa eliminar algumas das características que podem ser desnecessárias em um banco, como as padronizações gerais, a fim de se obter um desempenho maior em situações mais específicas.

Um banco de dados que se enquadra nessas características é o Cassandra. Ele está dentro de uma subcategoria dos bancos não relacionais chamada de bancos NoSQL e foi construído com seu uso voltado para grandes bases de dados e acesso remoto de alta velocidade [40]. Por não ser muito conhecido fora do ambiente acadêmico e de algumas organizações, e do número relativamente baixo de pesquisas sendo feitas com ele, este banco foi escolhido para ser usado nos experimentos desse trabalho, a fim de se obter sua performance diante de determinados tipos de dados.

O Cassandra, assim como boa parte dos bancos NoSQL, é voltado para um novo e mais moderno modelo de computação chamando de computação em nuvem. A computação em nuvem surgiu no final dos anos 90 em conjunto com algumas das mudanças que fizeram com que o fluxo de dados na *web* crescesse drasticamente [34]. Tais mudanças fizeram com que surgissem o termo *Big data* [24], dados que ocupam grande espaço e exigem certa velocidade de acesso durante o uso. Uma característica importante da computação em nuvem é visar a performance ao se trabalhar com *Big Data*.

A computação em nuvem é um modelo que permite acesso fácil, em todo lugar e sob demanda, de uma rede de recursos de computação configuráveis (como exemplo: redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente providos e fornecidos com o mínimo de esforço de gerenciamento ou interação com o provedor do serviço [46].

A seguir são expostos brevemente os tipos de dados a serem usados neste estudo de caso e os motivos para a escolha deles.

A Bioinformática surgiu pela necessidade de ferramentas computacionais para a análise de dados genômicos originados pelos sequenciadores dos projetos genoma. Com o crescimento nos estudos dessa área surgiu-se uma grande demanda por uma forma de

sequenciamento de mais baixo custo que estimulou o desenvolvimento de tecnologias de sequenciamento massivamente paralelos, produzindo milhões de sequências em uma só rodada [49]. Com o baixo custo uma quantidade muito maior de dados pode ser produzida em um tempo muito menor. No entanto, a atual preocupação não é mais somente com o processamento, mas também com o armazenamento e a busca dos dados produzidos, pois essa tarefa ainda gera muito custo.

Sendo assim têm-se uma grande quantidade de dados sendo produzida de forma rápida e estes dados têm como característica serem armazenados em grandes arquivos (da ordem de GB) e tem-se uma dificuldade quanto à persistência destes, podendo assim ser considerados *Big Data*.

Para isso, a computação em nuvem e seus bancos NoSQL podem ser aplicados nesses testes. Com boas condições de terem resultados satisfatorios.

1.1 Problema e Hipótese

Como problema existe a dificuldade no armazenamento do grande volume de dados gerados por sequênciadores genéticos. A hipótese dada é de que o banco de dados em nuvem Cassandra é adequado para a persistência de tais dados.

1.2 Motivação

A importância deste estudo de caso é válida tanto para a área de bioinformática quanto para a computação, pois seu desenvolvimento pode permitir:

- Uma alternativa ao armazenamento dos dados da bioinformática.
- Uma análise prática em uma área nova dos bancos de dados que são os Bancos NoSQL.

1.3 Objetivo

O principal objetivo desse trabalho é o estudo do comportamento do banco de dados NoSQL Cassandra com dados da bioinformática, buscando desenvolver uma implementação adequada e verificar se é uma opção viável no aspecto performance.

1.3.1 Objetivos Específicos

Para alcançar o objetivo geral do trabalho, os seguintes objetivos específicos foram desenvolvidos:

- Definir uma metodologia para criar e manter um ambiente de nuvem do Cassandra. Isso inclui a escolha do sistema operacional, da linguagem e das estruturas de dados a serem usadas.

- Fazer um estudo de caso com os dados da Bioinformática, testando a eficiência da inserção, busca e extração em um ambiente de nuvem do Cassandra. A partir disso gerar comparações entre o desempenho das consultas no Cassandra e na implementação relacional atual.
- A realização de testes variando o número de máquinas com o Cassandra instalado, verificando a variação no resultado das consultas e buscas a fim de se obter novos dados com esse acréscimo.

1.4 Estrutura do Trabalho

A estrutura desse trabalho é apresentada a seguir:

- Capítulo 2: Apresentação dos conceitos básicos de computação em nuvem, tais como definições e as características essenciais, além de modelos e diferentes classificações que esses ambientes computacionais podem ter.
- Capítulo 3: É feita uma exposiçao das características e limitações dos bancos relacionais e dos bancos não relacionais NoSQL. Isso é feito para comparar o Cassandra em relação aos outros bancos.
- Capítulo 4: Destinado a detalhar o Cassandra, mostrando sua origem, suas características gerais e seu funcionamento interno. Também são mostradas as suas limitações e algumas informações úteis de configurações dele.
- Capítulo 5: Apresentação de como foi criado e implementado o ambiente dos testes. É mostrando como são os dados de entrada e o ambiente de nuvem criado.
- Capítulo 6: Exposição dos resultados atingidos seguidos de uma breve análise destes.
- Capítulo 7: Conclusões finais e possiveis trabalhos futuros.

Capítulo 2

Computação em Nuvem

Neste capítulo são abordados conceitos fundamentais relacionados à um ambiente de nuvem computacional, tendo por base o modelo definido pelo NIST (*National Institute of Standards and Tecnology*) [46]. Esse modelo é composto por cinco características essenciais, três modelos de serviço e quatro modelos de implantação. Na Seção 2.1, são apresentados os conceitos de computação em nuvem e como esse modelo é vantajoso para todas as partes envolvidas (fornecedor, desenvolvedor e usuário final) com o uso e fornecimento do serviço prestado. A Seção 2.2 tem como foco dois modelos computacionais: o *Grid Computing* e o *Utility Computing*, que são precursores da computação em nuvem que tornaram-na viável, são abordadas tantos as vantagens quanto as desvantagens destes dois modelos. Na Seção 2.3, as características adotadas pelo NIST como essenciais para que um ambiente seja caracterizado como computação em nuvem são descritas [53]. A Seção 2.3 apresenta os três modelos de serviço que o ambiente de computação em nuvem pode prestar. Por fim, as questões referentes ao modelo de acesso e disponibilidade do serviço, chamados de modelos de implantação [46], são abordados.

2.1 Definições de Computação em Nuvem

Vários autores criaram sua própria definição de computação em nuvem [1, 53], algumas destas podem ser vistas na Tabela 2.1, porém uma das definições mais aceitas é a descrita pelo NIST que diz: "Computação em nuvem é um modelo que permite um acesso fácil, em todo lugar e sob demanda de uma rede de recursos de computação configuráveis (a exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente providos e fornecidos com o mínimo de esforço de gerenciamento ou interação com o provedor do serviço" [46].

Trata-se de uma abstração que oculta do usuário final uma estrutura compartilhada para a execução de um serviço, na qual este usuário não se preocupa com o funcionamento interno do serviço, a disponibilidade, ou até mesmo se há risco de perda dos arquivos, tudo isto fica a cargo de quem presta o serviço de computação em nuvem [53]. Como é visto na Figura 2.1 algumas características críticas presentes em uma nuvem computacional só podem ser acessadas pelo desenvolvedor e o usuário final tem acesso somente à aplicação.

As vantagens da computação em nuvem, descritas brevemente a seguir e mais detalhadas na Seção 2.3, afetam todas as partes ligadas à este modelo; o usuário final, a empresa que aloca uma aplicação na nuvem e a empresa que provê o serviço de computação em

nuvem. O usuário final passa a ter acesso a uma aplicação, a um serviço, ou apenas a documentos de forma simples, sem ser necessário grande conhecimento na área de tecnologia, nem mesmo uma plataforma de hardware com grande poder de processamento ou armazenamento, a única necessidade é uma conexão com a internet. As vantagens para o usuário também passam pela privacidade, pois o serviço normalmente só é prestado aos usuários que possuem o código de segurança, passam pela flexibilidade, seja na troca de um hardware ou até mesmo na recuperação do conteúdo em caso de furto do aparelho, e passam pelo ganho econômico, pois não se faz mais necessário a troca de hardware para acompanhar a tecnologia [1].

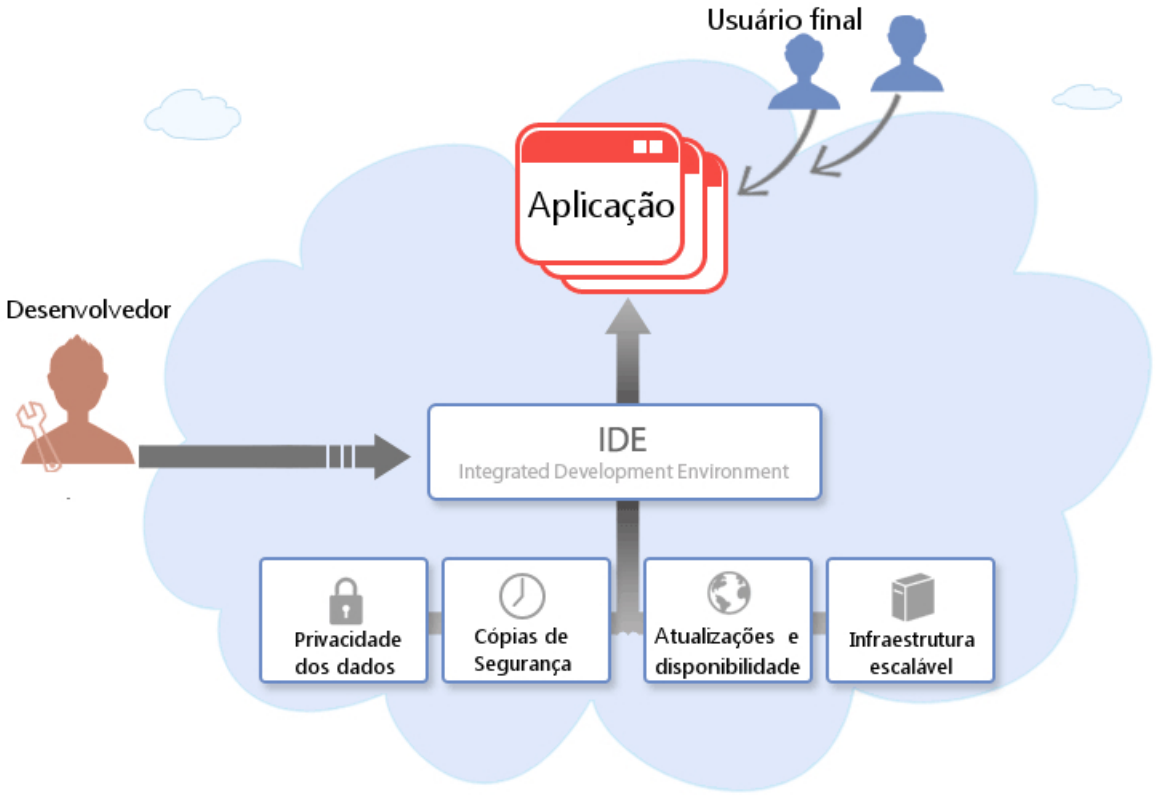


Figura 2.1: Modelo de *cloud*, adaptado de [14]

Uma vez que o armazenamento e o processamento de uma aplicação é terceirizado, a empresa que contrata a alocação na nuvem pode dispor de mais espaço físico. Outra vantagem ao se alocar uma aplicação em um ambiente de computação em nuvem é que não existe uma preocupação com a atualização nem com a manutenção dos hardwares o que faz com que sejam reduzidos os gastos computacionais [53], além de não ter que contratar funcionários extras para dar suporte a esta área, ficando somente focada no desenvolvimento e atualização da aplicação.

A prestadora do serviço de computação em nuvem tem a responsabilidade na atualização dos hardwares para que o processamento sempre esteja disponível e também na redundância das informações armazenadas, evitando que conteúdo se perca caso exista uma falha. Para uma empresa que possui o ambiente e a aplicação em nuvem a vantagem é a relação com o usuário e a uniformidade do serviço prestado, pois tal serviço será o mesmo independente do hardware utilizado pelo usuário.

Tabela 2.1: Definições de *Cloud*, tabela adaptada de [57, 10]

Autor/referência	**Ano**	**Definição**
R. Buyya	2008	Uma nuvem computacional é um tipo de sistema distribuído e paralelo que consiste em um conjunto de computadores interconectados e virtualizados que são providos dinamicamente e presentes como uma ou mais fontes de recursos baseados em contratos de serviços estabelecidos através de uma negociação entre o provedor do serviço e o usuário.
R. Cohen	2008	Computação em nuvem tenta abranger uma variedade de aspectos ligados ao desenvolvimento que vão desde a implantação, balanceamento de carga, fornecimento de recursos, modelo de negócio e arquitetura (como Web 2.0). É o próximo passo lógico em software. A mais simples explicação para a computação em nuvem é descrevendo-a como "software centrado na internet".
J. Kaplan	2008	Computação em nuvem é uma ampla variedade de serviços baseados na *Web* que visa permitir que os usuários obtenham uma gama de capacidades funcionais no formato "pague o que usar", que antes exigiam investimentos enormes em hardware/software e habilidades profissionais para adquirir. A computação em nuvem é a realização dos ideais anteriores de *Utility computing* sem as complexidades técnicas ou preocupações de implantação.
A. Ricadela	2008	Projetos de computação em nuvem são mais poderosos e resistentes a falhas do que sistemas de *Grid*, mesmo os que foram desenvolvidos recentemente.
I. Wladawsky Berger	2008	O principal aspecto é virtualizar ou esconder do usuário a complexidade, todos os softwares (ao longo do tempo) serão virtualizados, sendo gerenciado pelos sistemas e/ou profissionais que estão em outro lugar - na nuvem.

2.2 *Grid Computing* e *Utility Computing*

A computação em nuvem não foi a primeira ideia de uma aplicação ou serviço que estivesse armazenado em um ambiente distribuído. Mladen A. Vouk chega até mesmo a afirmar que a computação em nuvem apenas é uma junção de áreas como: a virtualização, computação distribuída, *Grid computing*, *Utility computing*, redes e serviços de software [58]. Assim para se ter um bom entendimento do que se trata a computação em nuvem é necessário conhecer *Grid computing* e *Utility computing* que são seus precursores principais.

2.2.1 *Grid Computing*

Grid computing é uma arquitetura orientada a um serviço ou aplicação [58], pode ser definido como um sistema distribuído de nós heterogêneos, separados geograficamente em diversos domínios administrativos, baseado em sua viabilidade, capacidade, performance, custo e requisitos dos usuários quanto à qualidade de serviço [51].

A *Grid* surgiu em um momento em que os recursos físicos eram muito caros, não era acessível um computador de alto processamento, começando então a surgir os grandes *data centers*. Esse poder de processamento era usado para um número limitado de aplicações, geralmente criadas pelos próprios usuários desses sistemas [45], que tinham geralmente objetivos científicos de cálculos em grande quantidade ou que fosse necessária alta precisão. Cada aplicação diferia muito uma da outra e seria impossível ter recursos de hardware suficientes que servissem para cada aplicação, seria necessário um *data center* para cada uma delas.

Em meados dos anos 80 surgiu a ideia de agregar essas aplicações dentro de diversos *data centers* de forma que cada uma utilizasse o processamento de diversos *data centers*. Um *data center* comportaria vários serviços e cada serviço se utilizaria de diversos *data centers* esse é um modelo conhecido como *network-centric computing* [45]. No início dos anos 90 muito foi investido em termos de conhecimento para que a *Grid computing* crescesse, a meta era de dar a ilusão de um recurso sempre disponível e de fácil acesso, como a energia elétrica, por isso foi dado esse nome (uma alusão a *"eletrical grid"*). Diversos projetos foram construídos a partir desse modelo, alguns são:

- Projeto Condor, desenvolvido em 1984 na Universidade de Wisconsin, com a meta de fazer uma distribuição de recursos entre todos os computadores de um departamento da universidade [56];
- Projeto Codine, desenvolvido por volta dos anos 90, provia uma interface gráfica simples para observar todos os recursos disponíveis. Era um projeto de código aberto em parceria com a empresa Sun Microsystems. Esse projeto foi muito similar ao Condor e a próxima geração do Codine se tornou o *Sun Grid Engine* e era disponível gratuitamente na Internet [35];
- Projeto *Legion*, desenvolvido em 1993 na Universidade de Virginia, tinha uma abordagem voltado à orientação de objetos. No entanto muitos serviços para ambientes distribuídos não eram orientados a objeto [36];
- Projeto Nimrod/G, desenvolvido em 1994 na Universidade de Monash, Austrália, com o objetivo de distribuir recursos para serviços heterogêneos [2, 9];

O modelo de *Grid computing* proposto trazia grandes benefícios, seria possível ter acesso aos recursos de hardware que estavam geograficamente distribuídos em diversas organizações. Ele dava um acesso transparente e instantâneo ao usuário da aplicação. Provia recursos extras para resolver problemas antes não solucionáveis devido à falta de recursos.

Foi criada uma infraestrutura mais confiável em que existia a possibilidade de agregar recursos sob demanda, em vários locais. Isso para satisfazer uma necessidade de recursos que não podem ser imprevistos. Para esse modelo funcionar corretamente explorava-se os recursos que eram pouco ou até mesmo não utilizados, pois, de outra forma, seriam desperdiçados. Essas vantagens também afetaram o campo da manutenção do hardware, pois diminuiu o esforço da gestão se comparado a múltiplos computadores com múltiplos sistemas independentes [60].

Por fim, o *Grid computing* também traz uma redução de custos com novos processadores, memória e locais para armazenamento de dados, uma redução do tempo de resolução de um processo, pela maior quantidade de recursos e a consequência é um aumento da produtividade [51].

Na época do desenvolvimento do *Grid computing* os serviços utilizados na *Internet* não distinguiam quais lugares tem mais ou menos processamento para distribuem esses recursos. A ideia do *Grid computing* era realocar esses recursos, até mesmo agregar, para um máximo ganho no desempenho e para solução de problemas complexos que trariam grandes vantagens. Porém olhando em retrospecto é possível perceber que o modelo teve diversas desvantagens fazendo com que o *Grid* não tivesse o reconhecimento esperado, só voltando a tona recentemente quando a *Cloud computing* incorporou algumas características dele. Algumas dessas desvantagens são [58, 51]:

- Os desenvolvedores eram difíceis de encontrar, por ser uma tarefa complexa, era preciso que os encarregados fossem muito capacitados. Os desenvolvedores eram responsáveis também pela manutenção e era necessário que eles fossem especializados nas áreas de redes, hardware, armazenamento, programação de baixo nível, sistemas operacionais, entre outras.
- Outra dificuldade do *Grid* é que não existia um controle formal de quantos recursos cada aplicação poderia utilizar para solucionar seu processamento, podendo fazer com que uma aplicação demande tantos recursos quanto possível e prejudicando o processamento de outras aplicações.
- Uma das principais desvantagens foi a heterogeneidade dos recursos de hardware que são dispostos, pois isso que desencadeia a dificuldade na manutenção do hardware e no controle da distribuição destes.
- Outra desvantagem foi que o *Grid* foi criado para ser voltado somente para estudos científicos e a heterogeneidade dos domínios administrativos(diferentes universidades e órgãos públicos) prejudicou o avanço dessa tecnologia.
- As aplicações do *Grid* não eram de fácil usabilidade, não havia tanta preocupação com a interface do software e o usuário final.

Além de trazer essas desvantagens citadas, o ambiente em que esse modelo foi criado não foi propício para seu desenvolvimento. Desde o surgimento da *Grid* houveram diversas

transformações tecnológicas, nas quais o computador pessoal se desenvolveu tanto quanto os científicos e o investimento voltou-se para esses, visto que possuiam mais público, o que fez com que não se prolongasse o desenvolvimento de uma computação distribuída a níveis de pesquisa, mas sim para o usuário final. Apesar destas desvantagens algumas características como compartilhamento de recursos e uma arquitetura voltada a uma aplicação foram aprimorados e incorporados pela *Cloud Computing*.

2.2.2 *Utility Computing*

Os estudos feitos utilizando o *Grid computing* deixaram claro que o processamento computacional seria mais eficaz se feito de forma distribuída em uma rede de computadores. Em busca de estabelecer redes de recursos distribuídos e oferecer o acesso a estes recursos como serviço foi criado o conceito de *Utility computing*. O *Utility computing* foi fortalecido com o pensamento de Gordon Bell o qual sugeriu em uma conferência em 1992 o uso da programação paralela em massa [60, 45]. O *Utility computing* foi semelhante ao *Grid*, porém agora um modelo que não era voltado somente as pesquisas científicas. O *Utility computing* tem por objetivo prover um mecanismo de compartilhamento que torna mais eficaz o uso dessa infraestrutura de recursos computacionais para o usuário apenas quando requisitado por esse. Os recursos que tem um potencial subutilizado podem ser realocados de forma que essa subutilização seja mínima [4].

Utility computing pode ser dividido em duas categorias: o modelo de recursos de um servidor completo (*full server utility model*) [4] em que os recursos são adquiridos e liberados sob demanda e o modelo de recursos distribuídos (*shared utility model*) [4] no qual uma quantidade limitada de recursos é explorada por múltiplos usuários ao mesmo tempo. O modelo de recursos distribuídos pode ser uma aplicação dentro de um modelo de recursos de um servidor completo. No modelo de servidor completo são usados mecanismos de escalonamento, porque há uma grande quantidade de requisições e o controle da aplicação tem de ser preciso ao alocar e liberar recursos.

Utility computing oferece vantagens tanto ao provedor como para o usuário. Os usuários pagam apenas pelo uso do processamento, armazenamento e comunicação entre uma fonte e outra, além do que o provedor entrega um serviço que distancia o usuário final da complexidade do hardware, ele não precisa mais se preocupar com manutenção, nem mesmo com aquisição de novos componentes para melhorar sua performance ou apenas se equiparar com a tecnologia utilizada [50]. O grande gasto com novos *harware* se torna um pequeno custo por um serviço que se encaixa perfeitamente em seu perfil, sem perda ou sobra de recursos computacionais. Já pelo lado do provedor não é necessário ter um tratamento exclusivo para cada usuário, uma solução virtualizada é configurada para que, de forma dinâmica, satisfaça um grupo de pessoas. Sendo essa alocação de recursos dinâmica fica fácil para o provedor realocar recursos para os usuários com maiores prioridades e isso faz com que a chance de existirem recursos subutilizados se tornem mínimas [4].

Como é possível observar o *Utility computing* caracteriza-se por um serviço voltado apenas para a infraestrutura de uma aplicação, que busca segurança, flexibilidade, baixo custo com energia e escalabilidade. Com o tempo, a *Cloud Computing* se desenvolveu e o modelo *Utility computing* foi incorporado como um modelo de serviço da *Cloud Computing*, o *Infrastructure as a Service* que é tratado na Seção 2.3.

2.3 Características Essenciais da Nuvem

Para classificar uma solução como computação em nuvem, existem certas características que foram adotadas como sendo essenciais [53]. De modo que elas a definem entre os outros paradigmas e mostram as qualidades necessárias que um bom serviço de computação em nuvem deve ter. São elas [46]:

Self-service sob Demanda

O usuário adquire o recurso computacional unilateralmente, em que ele especifique o tamanho do espaço para armazenamento e o tempo de processamento. O hardware e o software dentro da computação em nuvem podem ser reconfigurados automaticamente, e essas alterações são informadas ao usuário. Podem ser criados perfis diferentes para, por exemplo, a instalação de softwares ou alterações nas configurações de rede.

Amplo Acesso

Recursos disponíveis por meio da rede seguindo padrões para acesso de diferentes dispositivos, como *notebooks* ou celulares. O acesso se faz com o uso de um navegador de internet ou algum outro software leve fornecido pela própria nuvem.

Pooling de Recursos

Para atender a diversos usuários, é criado um *pool* onde são agrupados os recursos computacionais do provedor. Eles são dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Os usuários não precisam saber exatamente a localização física desses recursos, apenas em um nível mais alto, como a localização geográfica aproximada (país, estado, *datacenters*). Os exemplos de recursos incluem: armazenamento, processamento, memória, largura de banda de rede e máquinas virtuais.

Elasticidade Rápida

Os recursos podem ser adquiridos ou removidos de forma rápida (automática ou não), se houver uma variação da demanda. Tal característica dá ao usuário a sensação de que existem recursos ilimitados que podem ser adquiridos a qualquer momento e em qualquer quantidade. Normalmente essa característica é alcançada fazendo-se uso de uma máquina virtual que permite sua reconfiguração durante a execução.

Serviço Medido

O uso de recursos dos sistemas de computação em nuvem deve ser medido e analisado para conseguir uma otimização, feito de forma automática. Isso garante transparência para o provedor e para o usuário. A automação também é realizada no nível de abstração adequado para o serviço, como no armazenamento, processamento, largura de banda e contas de usuário. O uso de recursos pode ser monitorado, controlado e reportado, oferecendo transparência tanto para o usuário, quanto para o provedor do serviço utilizado.

2.4 Modelos de Serviço

Diferentemente do *Utility computing* que era um modelo voltado para a infraestrutura como um serviço, a computação em nuvem possui três modelos de serviço: o *Software* como Serviço (SaaS), a Plataforma como Serviço (PaaS) e a Infraestrutura como Serviço (IaaS) [53]. Tais modelos foram criados para se representar que tipo de padrão de arquitetura será usado em determinado ambiente. De acordo com o tipo de cliente ou de sua necessidade pode-se decidir o modelo mais adequado para se usar. Estes quatro modelos são melhores descritos a seguir.

SaaS

SaaS, *Software* como Serviço é o modelo que visa o usuário final. Trata-se da disponibilização de softwares com propósitos específicos acessados pelo usuário através da internet. Quando se fala em computação em nuvem fora do meio acadêmico ou empresarial, que não seja TI (Tecnologia da Informação), normalmente é a este modelo que as pessoas se referem.

Normalmente qualquer pessoa por meio de um ou mais dispositivos clientes com acesso a *web* pode fazer uso de seus recursos, através de uma interface simples e intuitiva. A computação em nuvem torna-se vantajosa ao usuário final, pois este desconhece e não tem poder para modificar a infraestrutura por trás do serviço, incluindo rede, servidores, armazenamento, ou o sistema operacional usado pelo serviço. Cabe a ele fazer uso do software e alterar apenas as configurações internas disponibilizadas.

O processamento da aplicação que é disponibilizada é descentralizado e o serviço ser prestado por meio da internet, podendo ser usado a qualquer momento e tomando poucos recursos da máquina, exigindo apenas um programa para se fazer o acesso, como um navegador ou a própria aplicação cliente. Outra vantagem é que devido a separação entre o usuário e o ambiente onde se encontra o software, é possível que se faça atualizações no sistema independente do hardware usado pelo usuário, a incorporação de recursos novos, atualizações ou qualquer tipo de evolução pode ser feito de maneira mais rápida. A notificação ao usuário informando tais mudanças é opcional.

PaaS

PaaS, Plataforma como Serviço, como o nome sugere, é uma plataforma de desenvolvimento para programadores implementarem aplicações em nuvem. Quem utiliza este serviço são os desenvolvedores, eles tem controle sobre as aplicações colocadas na infraestrutura e podem fazer uso de ferramentas disponibilizadas pela computação em nuvem para auxiliar o desenvolvimento.

Convencionalmente, é oferecido ao usuário o ambiente em que irá trabalhar, contendo um sistema operacional e o suporte para as linguagens de programação ofertadas.

O uso de PaaS pode ser visto como uma solução estratégica para empresas que querem terceirizar parte do processo de desenvolvimento, tornado sua equipe responsável apenas em usar as ferramentas e ambientes prontos em seus projetos.

IaaS

IaaS, Infraestrutura como Serviço é classificado como o modelo de mais baixo nível que pode ser oferecido em ambientes de computação em nuvem. Trata-se do espaço em si, em que serão colocados os dados, softwares, sistemas operacionais e qualquer tipo de aplicação. É fornecida ao usuário uma interface para a administração da infraestrutura.

Seu funcionamento consiste na virtualização de recursos computacionais, a fim de se garantir a possibilidade de alterações como o aumento ou redução de tais recursos de maneira dinâmica.

Na Figura 2.2 é mostrado como é feito o relacionamento entre os modelos e as pessoas que participam de alguma maneira do ambiente de computação em nuvem. Como pode ser observado o provedor fornece, pelo menos um, dos modelos de serviço da computação em nuvem. O desenvolvedor se utiliza do IaaS e do PaaS para forneçer o modelo SaaS. O usuário final somente consome o SaaS que pode ser tanto fornecido pelo provedor como pelo desenvolvedor.

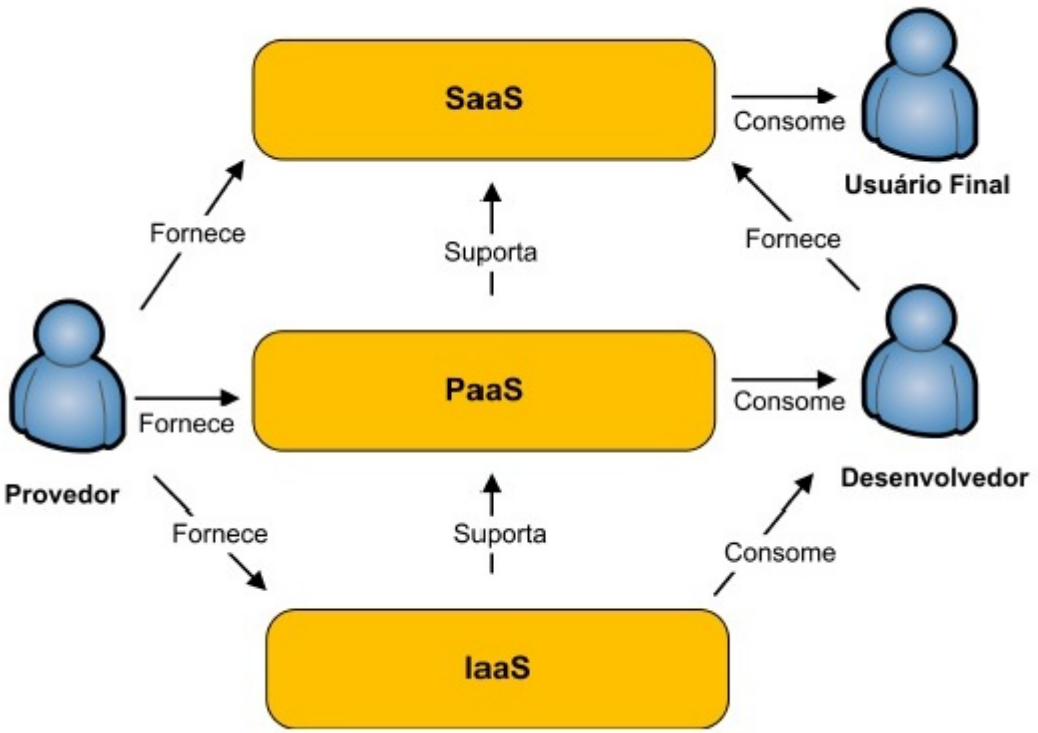


Figura 2.2: Modelos e papeis [53]

2.5 Modelos de Implantação

Outro tipo de classificação para ambientes de computação em nuvem é o seu modelo de implantação, referente ao acesso e a disponibilidade do serviço. As empresas podem possuir necessidades diferentes quanto a liberdade de uso de alguns de seus recursos.

Para resolver esse tipo de situação, surgiu a ideia de ambientes mais restritos, onde é exigida uma autenticação para se definir o nível do usuário e prover os serviços em que possui autorização.

Definem-se assim os modelos de implantação da computação em nuvem, que são quatro: nuvem privada, nuvem comunitária, nuvem pública e nuvem híbrida [46]. Cada um destes tipos é descrito a seguir.

Nuvem Privada

No modelo de nuvem privada, sua infraestrutura é guardada por um grupo ou organização, que é sua proprietária ou a terceirizou. Fazem uso de políticas de acesso para limitar seu uso interno em várias áreas, tendo cada uma algum privilégio diferente. No caso de usuários que não pertencem a esse grupo ou organização, o acesso não é permitido.

Nuvem Comunitária

Uma nuvem comunitária é compartilhada por várias organizações e suportada por alguma comunidade com objetivos em comum. Esse modelo de implantação pode ser mantido remotamente ou até localmente. Sendo mantido por uma das empresas envolvidas ou algum terceiro.

Nuvem Pública

Como o nome sugere, a nuvem pública é de acesso livre. Nesse modelo, qualquer pessoa, com conhecimento sobre o serviço, pode acessar a nuvem sem que haja qualquer restrição quanto as suas funcionalidades.

Nuvem Híbrida

Uma nuvem híbrida é uma composição de dois ou mais tipos de modelos de implantação. Costumam ser soluções únicas que seguem algum padrão estabelecido para fazer suas conexões e garantir a portabilidade.

Capítulo 3

Características e Diferenças entre Bancos de Dados Relacionais e Não Relacionais

Bancos de dados podem ser classificados como relacionais e como não relacionais. Neste capítulo, será apresentada uma breve introdução a bancos relacionais na Seção 3.1 e informações sobre bancos não relacionais voltados para NoSQL na Seção 3.2.

3.1 Bancos de Dados Relacionais

Um banco de dados é comumente definido como uma coleção de dados organizados e persistentes. A tarefa de armazenar os dados e controlar a sua estrutura a nível de hardware é do Sistema Gerenciador de Banco de Dados (SGBD). É esse programa que faz a interface entre os dados inseridos pelo usuário e o servidor onde os dados são armazenados.

A linguagem que o usuário deve usar para se comunicar com o banco é o SQL. Um linguagem de *script* declarativa fortemente baseada na álgebra relacional. Ela diferencia-se por sua simplicidade e pela facilidade de uso [39].

Todo banco de dados relacional é composto por tabelas que representam relações. Daí o nome relacional. Cada tabela é um conjunto não necessariamente ordenado de linhas ou tuplas. Essas linhas, por sua vez, compreendem uma série de campos onde estão guardados os valores dos atributos. Os atributos são associados às colunas da tabela.

Além das linhas não estarem ordenadas, também é possível diferenciar um banco de dados de um arquivo convencional pela possibilidade de consultar os dados usando qualquer critério adotado pelo usuário, as consultas podem ser feitas de modo a permitir encontrar conexões complexas entre as informações com apenas uma única busca [39].

No modelo relacional dois conceitos são fundamentais para o seu funcionamento de maneira correta: atributos chaves de uma tabela e as restrições de integridade.

Atributo Chave

Um conceito de grande importância para o entendimento das relações entre linhas de tabelas em um banco relacional é o conceito de chave, que podem ser classificada como chave primária e estrangeira.

A chave primária é uma coluna, ou combinação delas, cujos valores se distinguem de todas as linhas da tabela. Seu papel é, devido ao fato de ser única, localizar qualquer linha da tabela de maneira não ambígua. Como índice, ela também serve para representar uma entidade ao se fazer associações mais complexas [31].

A chave estrangeira tem o mesmo valor de uma chave primária, sendo utilizada para implementar relacionamentos entre tabelas. Seu uso serve para ser uma referência a um outro atributo, permitindo a implementação de dependências dentro do banco de dados. É dever do SGBD verificar se, quando uma chave primária é alterada, todas as chaves estrangeiras que fazem referência a ela também sejam atualizadas [39].

Restrições de Integridade

A integridade dos dados é um dos objetivos primordiais de um SGBD. Os dados de um banco são classificados como íntegros quando refletem de maneira correta o que estão querendo representar e são consistentes entre si. As restrições de integridade são regras que foram criadas como um mecanismo para garantir que essa consistência seja mantida. Para a abordagem relacional, elas podem ser classificadas segundo Heuser com as seguintes categorias [39]: integridade de domínio, integridade de vazio, integridade de chave e integridade referencial

Integridade de domínio são restrições que especificam que o valor de um campo tenha que seguir o tipo de dados que foi definido para aquela coluna. Não é permitido que o usuário final crie domínios novos para a sua aplicação, mas apenas use os domínios que já estão predefinidos (real, inteiro, caracteres, etc).

Integridade de vazio diz respeito a um campo poder receber o valor *null* e que seja impedido que o usuário quebre essa regra após defini-la. Os campos marcados como chaves primárias não podem ser deixados vazios.

A integridade de chave é a restrição que define que os valores de uma chave primária, como em um identificador (ID), devem ser únicos.

E por último, a integridade referencial define que os valores de um campo que pertence a uma chave estrangeira devem ser sempre iguais aos da chave primária que eles referenciam. Sendo feita uma alteração em uma chave primária, todos os valores das chaves estrangeiras referentes àquele atributo devem ser atualizados.

Todas essas restrições são de responsabilidade do SGBD, estando o usuário livre de ter que se preocupar com o comprimento dessas regras. Vale ressaltar que o usuário também pode criar suas próprias restrições de integridade conforme seus objetivos, sendo que no caso delas ele escreva explicitamente o código ou *script* que irá garanti-las [39].

3.1.1 Normalização

Um aspecto importante que deve ser lembrado ao se tratar de bancos de dados relacionais é a normalização. Trata-se do termo forma normal, uma regra formalizada que deve ser seguida para que uma tabela seja considerada "bem projetada". Existem várias formas normais, sendo cada uma delas com um conjunto de regras diferente, a fim de classificar o projeto dos bancos. São vistas aqui as três primeiras formas normais, que são as fundamentais para garantir um mínimo de redundância para um bom modelo de dados [31]. A Figura 3.1 mostra a comparação entre as formas normais no aspecto abrangência.

Primeira Forma Normal

Uma tabela encontra-se na primeira forma normal quando não contém tabelas aninhadas dentro dela, ou seja, informações não diretamente relacionadas guardadas de maneira redundante dentro da tabela.

Segunda Forma Normal

Uma tabela está na segunda forma normal quando, além de estar na primeira forma normal, não apresenta dependências parciais. Uma coluna não pode depender de apenas parte da chave primária, portanto quando uma tabela possui apenas uma chave primária e está na primeira forma normal, ela já está na segunda forma normal.

Terceira Forma Normal

Classifica-se uma tabela na terceira forma normal quando está na segunda forma normal e todo atributo não chave for dependente de outro atributo que não pertença à chave primária.

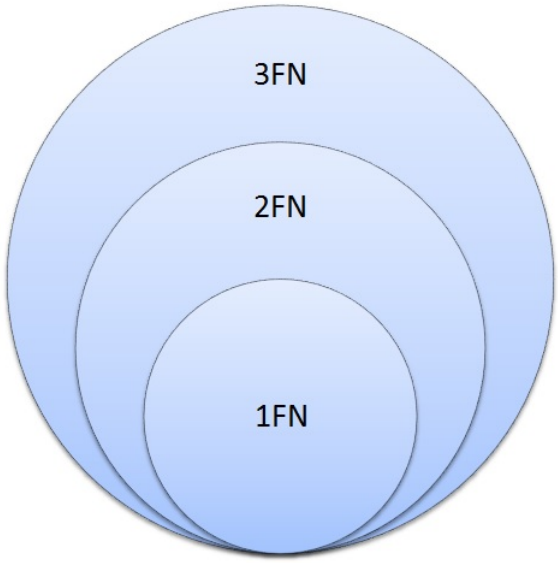


Figura 3.1: Hierarquia das formas normais [39]

3.1.2 Limitações

Os dados inseridos em um banco de dados relacional ficam em várias tabelas ligadas entre si por meio de chaves estrangeiras. O modelo relacional não obriga a criação de uma estrutura de tabelas coesa, assim programadores inexperientes podem projetar sistemas complexos sem necessidade, ou projetos que limitam o desenvolvimento futuro.

Com o aumento da complexidade do projeto do banco, torna-se mais difícil fazer a sua administração e, com o aumento das pessoas envolvidas, a possibilidade de erros ou inconsistências não pode ser desprezada.

Com o grande aumento do volume de dados, em alguns casos o desempenho de bancos relacionais já não é satisfatório devido ao crescimento do tempo de certas consultas [8].

Os bancos relacionais apresentam uma série de caracteristicas chamadas de ACID, que representam atomicidade, consistência, integridade e durabilidade dos dados. Apesar de ser um ótimo modelo, pode se tornar um obstáculo ao buscar-se uma maior liberdade de esquema ou mais recursos escaláveis [59].

3.2 Bancos Não Relacionais NoSQL

Os bancos de dados que não apresentam todas as características ACID são classificados como bancos não relacionais, como é o exemplo do modelo de banco de dados orientado a objetos [37, 3] e do NoSQL (*Not only Structured Query Language*). Como o Cassandra é um banco NoSQL esta seção focará nos bancos que seguem esse padrão.

Definição de NoSQL

O termo NoSQL foi usado pela primeira vez em 1998 para citar um banco de dados relacional *open-source* que omitia o uso de SQL, o Strozzi NoSQL, criado por Carlo Strozzi [34]. O nome veio pelo fato que o banco de dados não utiliza o SQL como a linguagem de busca, no lugar, o banco utilizava *scripts* de *shell*. Porém, o uso do termo como hoje é conhecido não se refere ao banco desenvolvido por Strozzi [34]. Em 2009, em uma conferência conhecida como *"NoSQL Meetup"*, que foi organizada por Johan Oskarsdon, o criador do Last.fm [55], o termo foi utilizado novamente, porém referenciando bancos de dados não relacionais. Nessa conferência ficou claro que o uso do Amazon's Dynamo e do Google Bigtable (precursores do movimento NoSQL) estava se popularizando, principalmente no meio das *start-ups* e um novo conceito estava se formando [55].

Uma definição de NoSQL é que ele é um conjunto de conceitos que permitem o processamento de dados de forma rápida e eficiente com um foco em performance [43]. É um modelo alternativo pensado para se modelar os dados sem ter que seguir os padrões rígidos criados para o modelo relacional. Como medida para se contornar esse problema, ele foi proposto trazendo consigo planos de eliminar ou reduzir essa estruturação [8]. Para evitar a perda de informações, o NoSQL tem uma arquitetura distribuída tolerante a falhas que se baseia na redundância de dados em vários servidores. Dessa forma o sistema pode ser escalado facilmente agregando mais servidores, e assim a falha de um deles pode ser tolerada.

3.2.1 Características do NoSQL

Bancos NoSQL são projetados para trabalharem com uma grande quantidade de dados distribuídos. Algumas características comuns em um banco de dados NoSQL são [43]:

- Prover uma grande escalabilidade horizontal. Essa característica consiste em cada aplicação ser capaz de aumentar seu número de nós com a camada IaaS, ou seja, é a grande capacidade de requisição de mais recursos de armazenamento e processamento.

- Prover uma baixa latência. Essa característica é obtida por meio da escalabilidade horizontal que se caracteriza pela otimização do tempo de um grande processamento dividindo ele em vários pequenos processos que são distribuídos em diferentes máquinas, assim o tempo de resposta diminui para uma ordem de alguns poucos milissegundos.
- Prover grande disponibilidade de acesso. Essa característica é obtida por meio do grande processamento que permite um acesso de um número de usuário maior que os gerenciadores de bancos de dados relacionais.
- Prover um processamento de grandes dados, seja uma grande quantidade de ciclos de leitura e escrita ou uma quantidade massiva de acessos de usuários.

Existem cenários em que o tempo de resposta é mais relevante que a consistência dos dados recebidos isso ocorre principalmente em serviços que provêm *streaming* de mídia (como músicas ou filmes online), páginas *web* com alto tráfego de usuários (redes sociais) e casos em que se indexam um grande número de documentos. Nessas situações, se a informação for recebida de forma atrasada para a proposta do serviço, já não será mais útil ou o grande número de acessos faria com que o processamento do banco não fosse eficiente.

Essas características estão entre as vantagens que fizeram com que o NoSQL se desenvolvesse rapidamente no meio empresarial e no meio cientifico quando lidam com um grande volume de dados e aplicações *web* em tempo real.

3.2.2 Modelos de Bancos NoSQL

No contexto de bancos de dados NoSQL é possível encontrar uma diversidade de modelos de dados, a Figura 3.2 ilustra um conjunto de modelos e aplicações NoSQL.

O armazenamento de dados do NoSQL, que está contido dentro do modelo não relacional, se difere do banco de dados relacional no aspecto que os sistemas de gerência de banco de dados relacionais armazenam apenas dados estruturados em várias tabelas, e os sistemas de gerência NoSQL armazenam tanto dados estruturados como dados não estruturados, dados estes armazenados em coleções sem relacionamentos entre si, assim são caracterizados como dados não estruturados. Por não possuírem relações geralmente não é possível fazer uma correlação entre essas coleções de dados por meio da linguagem de consulta. Para otimizar a consulta a forma que os dados são armazenados no banco é crucial, com esse fim foram elaborados alguns modelos de armazenamento de dados não estruturados: orientado a documentos, modelo chave/valor, tabular e orientado a grafos. A seguir são apresentados esses modelos. É importante lembrar que muitas aplicações não se encaixam somente em um modelo de dados e essa é uma lista que está em constante transformação.

Chave/Valor

O modelo de armazenamento chave/valor foi o modelo precursor do movimento NoSQL, pois a partir dos seus dois mais conhecidos expoentes, o Amazon DynamoDB e o Google BigTable, que foi observado, em 2009, que os bancos de dados não relacionais poderiam ser uma alternativa aos modelos SQL [38].

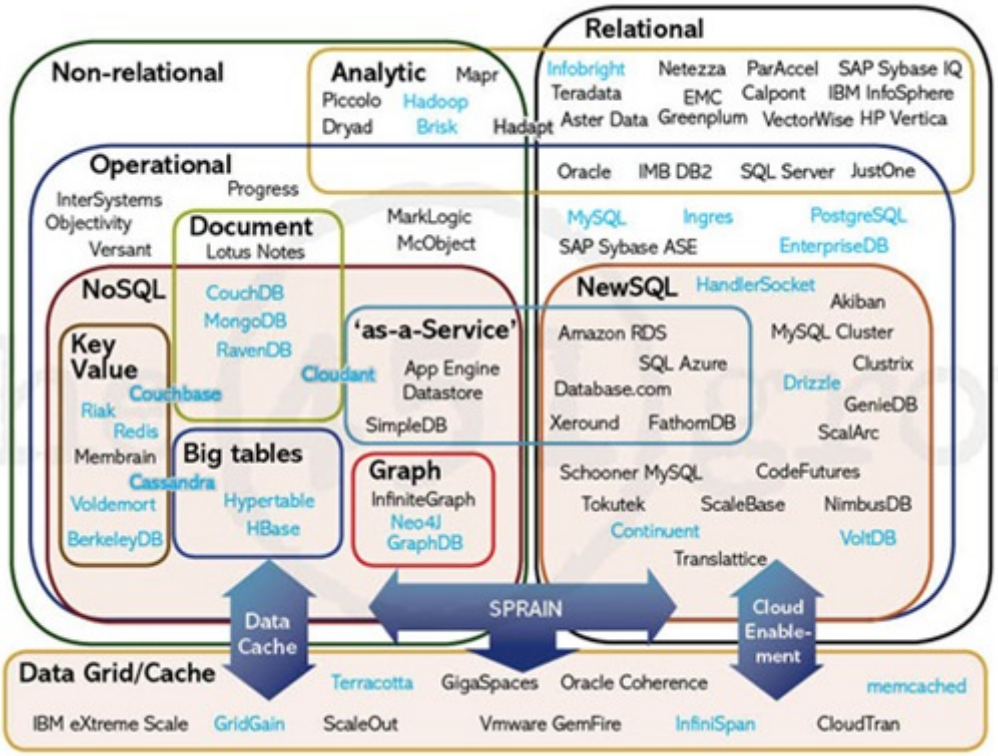


Figura 3.2: Aplicações e organização por modelos de dados [54]

O armazenamento chave/valor é similar ao mapeamento feito em dicionários e enciclopédias. Os dados são endereçados por uma única chave. Tendo os valores como apenas uma sequência de *bytes* que não apresentam significado, o sistema não os interpreta ou os classifica. Portanto a única maneira de guardá-los é através de chaves para cada valor armazenado. Por consequência, os valores são isolados e completamente independentes um do outro, tornando necessário que as relações sejam tratadas pela lógica de aplicação.

Devido também a essa estrutura simples, o banco de dados é livre de qualquer esquema. Novos valores podem ser inseridos a qualquer momento sem que haja conflitos com os dados já guardados e não afetando a disponibilidade do sistema. O agrupamento chave/valor em coleções é o único meio de se acrescentar uma estrutura ao banco de dados.

Este modelo é útil para se fazer operações simples, que se baseiam em atributos indexados. Uma vantagem desse modelo é a velocidade de suas consultas, especialmente as usadas com mais frequência [38].

Documento

Os bancos de dados que seguem esse modelo têm como o principal conceito os documentos. Eles os armazenam e os recuperam. Esses documentos são auto-descritivos, são estruturas de dados de árvores hierárquicas que podem representar mapas, coleções, e outros objetos. Os documentos armazenados ali apresentam uma similaridade uns com os outros, mas não têm de ser exatamente a mesma formatação.

Esses bancos de dados assemelham-se aos bancos chave/valor, tanto que são considerados por muitos como o próximo passo de um armazenamento chave/valor simples para estruturas de dados um pouco mais complexas e significativas [55]. Isso se dá pelo fato dos bancos de dados baseados em documentos armazenarem os documentos na parte do valor do armazenamento de chave/valor. É possível compreender os bancos de dados de documentos como bancos chave/valor nos quais é possível se pesquisar dentro de um campo. Dentro dos documentos, as chaves devem ser únicas. Para cada documento existe um identificador chamado ID que também deve ser único e serve para identificar o documento na coleção.

A diferença entre este modelo e o modelo chave/valor, é que no modelo de documentos os valores não são ocultos para o sistema e podem ser buscados ou referenciados também. Assim, estruturas mais complexas como objetos aninhados podem ser tratadas com mais conveniência. Uma outra vantagem de guardar dados em documentos JSON é o suporte para tipos de dados, tornando o uso mais fácil para desenvolvedores.

Assim como o modelo chave/valor, o modelo de documentos é livre de restrições de esquema. Isso facilita inserir novos documentos ou atributos aos já existentes durante a execução.

Uma das várias vantagens desse modelo é a facilidade de migração e integração de bases de dados. A organização dos atributos pode permitir também uma facilidade maior de gerar estatísticas [38]. Os bancos mais conhecidos baseados em documento são o MongoDB, desenvolvido pela 10gen, *open-source*, e o CouchDB, desenvolvido pela Apache [34].

Orientado a Colunas

O modelo orientado a colunas, também conhecido como modelo tabular, é um modelo desenvolvido para suportar uma quantidade muito grande de dados. Trata-se de um mapa de dados amplo, persistente, distribuído e multidimensional. Como as informações não são interpretadas pelo banco de dados, elas podem ter tamanhos variados e, por consequência, devem ser tratadas e organizadas pela camada de aplicação. Múltiplas versões de um valor são armazenadas em ordem cronológica para se ter suporte a versionamentos e a possibilidade de se comparar a performance entre elas.

As colunas podem ser organizadas em famílias de colunas para facilitar a organização e o particionamento do banco, podendo também ser adicionadas a qualquer momento. Entretanto, uma família de colunas não pode ser definida durante a execução, o que leva a uma menor flexibilidade em relação aos modelos chave/valor e documento [38].

O banco de dados Cassandra é uma implementação do modelo tabular, porém com um conceito a mais, chamado de "super-colunas", que permite ao Cassandra a possibilidade de trabalhar com estruturas de dados mais complexas. Mais informações sobre o Cassandra e sobre esse modelo de dados são apresentadas no Capítulo 4.

Orientado a Grafos

Diferente do modelo relacional e dos modelos não relacionais vistos até agora, o modelo orientado a grafos se especializa no gerenciamento eficiente de dados fortemente conectados. Aplicações baseadas em dados com muitas relações são mais adequadas para esse modelo de banco devido ao custo de fazer buscas com muitas conexões.

Como o nome sugere, esse modelo permite que o banco de dados apresente a forma de um grafo. Conforme é definido em um esquema, o sistema constrói um grafo onde cada nó possui uma par chave e valor. Uma aplicação deste modelo é para se encontrar um caminho em sistemas de navegação, pois possui a facilidade de deslocar ao longo dos nós com eficiência [38].

3.2.3 Limitações

Como apresentado nesse capítulo, o NoSQL é um gerenciador de banco de dados flexível e muito poderoso para armazenar e buscar diversos tipos de dados, ainda assim ele possui suas limitações. O desenvolvimento de bancos de dados relacionais tem de satisfazer os quatro requisitos básicos estipulados pelo padrão ACID, que consistem em: atomicidade, consistência, isolamento e durabilidade.

Recentemente, em 2000, Eric Brewer propôs, em um simpósio, o teorema CAP, ou Teorema de Brewer, que elucida as limitações de sistemas escaláveis e, consequentemente, do NoSQL. As três principais características segundo o teorema CAP são [34]: consistência dos dados (*Consistency*), disponibilidade (*Availability*) e tolerância ao particionamento (*Partition Tolerance*).

- Consistência: propriedade que dita como e quando o sistema está em uma versão consistente após a execução de uma operação. Um sistema distribuído é considerado consistente se todos os usuários leitores têm a mesma visão de uma atualização feita por um usuário que escreve no sistema, logo após essa atualização ser efetuada.
- Disponibilidade: propriedade na qual um sistema é projetado e desenvolvido que permita ao usuário ler e escrever a qualquer momento no banco de dados.
- Tolerância ao particionamento: característica em que um sistema pode executar corretamente, apesar da divisão física da rede. Também pode ser compreendido como a habilidade de um sistema de lidar a adição e remoção dinâmica de nós de recursos. Entende-se que um sistema com alta tolerância ao particionamento é um sistema altamente escalável.

No entanto o teorema determina que em um sistema facilmente escalável só é possível garantir apenas duas destas três características, pois elas tendem a se anular. Desta forma, pode-se concluir que ao se ter disponibilidade e tolerância ao particionamento é impossível ter consistência, pois um usuário poderia escrever e ler a qualquer momento, mas dessa forma o banco de dados não está sempre conectado à versão pessoal do usuário. A solução para esse caso é fazer com que o usuário tenha somente sua aplicação atualizada, qualquer aplicação que é compartilhada não tem acesso a última versão dos dados.

Da mesma maneira, ao se dar a liberdade do usuário ler e escrever a qualquer momento e tendo como prerrogativa que todos os usuários estarão vendo uma versão consistente do sistema, não é possível permitir que a rede tenha uma divisão física, pois a divisão física quebraria o princípio da consistência.

Por último ao se ter consistência e tolerância ao particionamento é impossível que o usuário escreva e leia a todo momento. Ao se dar essa liberdade de leitura e escrita o sistema distribuído não tem como se manter estável para todos que o veem. A solução é

o usuário fazer uma requisição ao banco e aguardar a resposta. A Figura 3.3 esquematiza esse comportamento, também são mostrados nela exemplos de bancos de dados que pertencem a cada grupo de características.

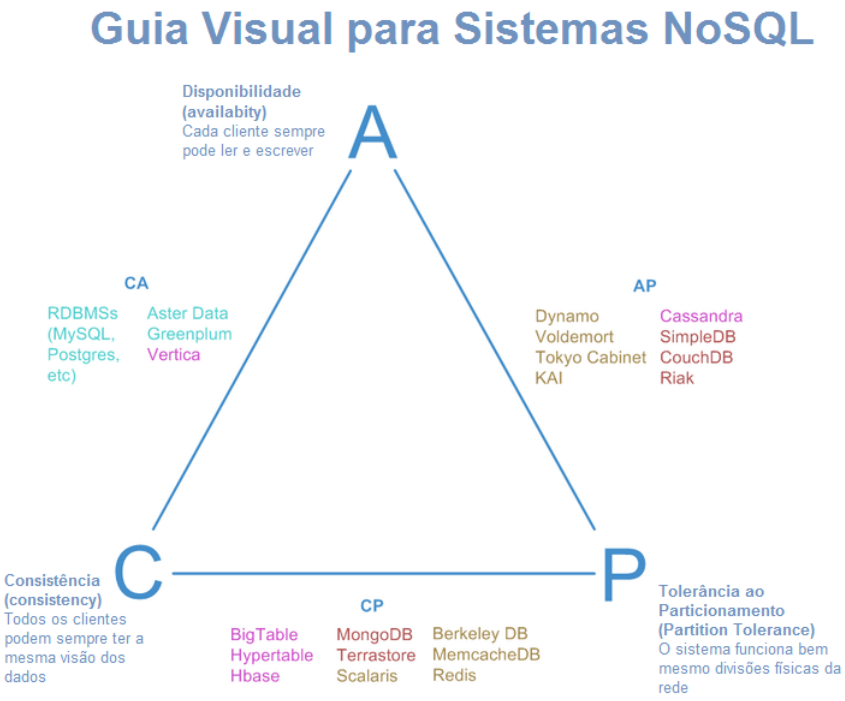


Figura 3.3: Aplicações e organização por modelos de banco de dados. Adaptada de [42]

Capítulo 4

Cassandra

Como dito em capítulos anteriores, o banco de dados NoSQL usado nesse trabalho foi o Cassandra. Este capítulo descreve a definição deste banco de dados, assim como características e funcionalidades pertinentes ao estudo de caso que foi realizado. A Seção 4.1 trata do modelo de dados do Cassandra e como o banco interpreta os dados coletados dos servidores. A Seção 4.2 aborda a arquitetura do Cassandra, como é feito o armazenamento do banco na estrutura física. Em específico a Seção 4.2.1 trata de como é feita a distribuição e replicação dos dados do Cassandra, a Seção 4.2.2 descreve os três principais particionadores do Cassandra, a Seção 4.2.3 explica como é possível ajustar a consistência dos dados. Na Seção 4.3 são vistos alguns problemas que podem acontecer durante o uso do Cassandra.

4.1 Definição e Modelo de Dados

Segundo seus criadores, Avinash Lakshman e Prashant Malik, o Cassandra é um banco de dados distribuído massivamente escalável, criado para armazenar uma grande quantidade de dados espalhados por vários servidores e, ainda assim, oferecer alta viabilidade de acesso e dados consistentes [44]. Seu lançamento ocorreu em 2008 como um projeto *opensource* desenvolvido e utilizado pelo Facebook para melhorar seu mecanismo de pesquisa [44]. Em 2009 foi adotado pela Apache Software Foundation [25] e hoje é usado por grandes empresas como Netflix [15, 25] e Ebay [48], além de órgãos governamentais como a NASA [47, 11].

Em seu desenvolvimento, baseou a arquitetura de seu sistema distribuído na arquitetura do Dynamo da Amazon, enquanto seu modelo de dados é baseado no BigTable do Google [44]. Apesar do seu modelo de dados voltado a coluna, o Cassandra permite a consulta como o modelo chave-valor, característica que foi adotada do Dynamo [32], podendo assim ser considerado como um modelo híbrido.

4.1.1 Características Gerais

Dentre as principais características que tornam o Cassandra um banco de dados vantajoso em diferentes situações, pode-se listar [40]: distribuído, descentralizado, escalável, alta disponibilidade, tolerante a falhas, consistência e alta performance. Segue uma descrição de cada um destes atributos.

Distribuído e Descentralizado

Ele é capaz de executar em múltiplas máquinas, enquanto para o usuário aparenta estar executando em apenas uma. Isso é usado para o aumento da performance ao se fazer operações paralelas e para conseguir uma maior segurança devido a redundância de dados.

Diferente de outros bancos de dados, em que os módulos são separados entre *mestre* e *escravos*, no Cassandra, cada módulo possui igual importância. Isso é chamado de simetria de servidor e é um dos fatores que torna a disponibilidade do sistema alta.

Escalabilidade Elástica

Escalabilidade é uma característica que faz o sistema manter a performance mesmo com o aumento do número de requisições. Mais máquinas podem ser adicionadas para executar o processamento sem prejudicar o funcionamento do sistema, sem a necessidade de reiniciar algum processo ou editar os comandos.

Essa característica também se refere ao caso de remover máquinas do sistema sem que ele pare de funcionar.

Disponibilidade e Tolerância a Falhas

De modo geral, a disponibilidade de algum sistema está na sua capacidade de completar requisições. Sempre existe a possibilidade de ocorrer um erro de hardware ou de que dados sejam corrompidos em algum momento de sua transmissão. O Cassandra busca reduzir as chances da ocorrência desses erros fazendo uso de redundâncias e realocação de recursos quando apresentam chances de falha. Entretanto, mesmo se ocorrer um erro, ele irá dar continuidade a um procedimento a fim de obter um resultado final, apesar de estar parcialmente correto. Essa é uma medida para evitar que todo um projeto se comprometa devido a pequenos erros.

Consistência

Consistência refere-se essencialmente a capacidade de que, sempre que for feita uma leitura, o dado lido está na sua versão mais recente. Apesar de parecer algo trivial, trata-se de uma característica difícil de conseguir em sistemas distribuídos entre vários servidores. como o Cassandra.

O Cassandra faz uso de mecanismos para garantir essa consistência mesmo quando ocorrem operações paralelas, entretanto isso reduz a disponibilidade do sistema. Esse conflito deve ser resolvido pelo usuário, pois lhe foi conferida a possibilidade de definir qual será o grau de consistência de sua aplicação.

Alta Performance

O banco de dados Cassandra foi desenvolvido para usar multiprocessamento e tirar vantagem disso. Ele pode escalar por centenas de *terabytes* mantendo a consistência. Seu uso é recomendado em ambientes que fazem uso dessas variações e precisam de alta performance para poder ter acessos rápidos.

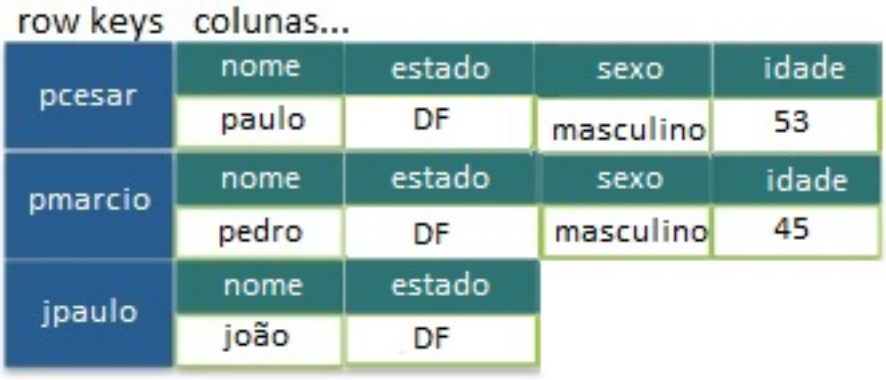
4.1.2 Modelo de Dados

O Cassandra é um banco não relacional, portanto seu modelo de dados difere do modelo relacional pois não é focado nas tabelas e relacionamentos entre essas, em vez disso ele é mais focado no desempenho das consultas a serem feitas. Assim, não existe uma estrutura que estabelece os relacionamentos entre uma tabela e outra. Como dito, seu modelo de dados era muito semelhante ao do BigTable, inclusive ambos ainda possuem o mesmo conceito para famílias de colunas [12], porém o modelo do Cassandra já sofreu alterações desde a sua primeira distribuição.

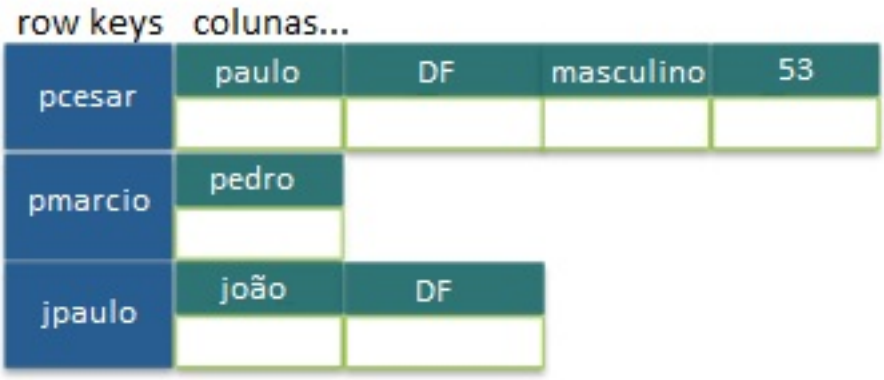
Inicialmente, o modelo de dados era constituído por três estruturas: as super colunas, famílias de colunas, colunas e linhas. Cada família de colunas possuía um conjunto de colunas, já as super colunas continham um conjunto de colunas ou um conjunto de famílias de colunas [12]. Recentemente, na versão 1.1, as super colunas foram descontinuadas devido a importantes limitações como não ser possível o uso de uma chave estrangeira em uma super coluna, e por performance, pois para a operação de leitura que qualquer coluna contida em uma super coluna era necessário carregar toda a super coluna para a memória [17]. Na versão 1.2 do Apache Cassandra, os elementos essenciais do modelo de dados são os *Keyspaces*, as famílias de colunas, as tabelas, colunas e linhas, que são descritos a seguir [27]:

- *Keyspace* é o agrupamento de dados que se assemelha a um esquema em um banco de dados relacional, porém o *keyspace* também possui informações sobre como os dados serão replicados ao longo do *cluster* (estrutura física da nuvem que será abordada ainda neste capítulo). Toda a família de colunas deve estar dentro de um *keyspace*, geralmente é criado um *keyspace* por aplicação.
- As famílias de colunas ou apenas tabelas são um agrupamento de colunas ordenadas por nome que é pesquisada por linha (uma característica de um modelo chave-valor). Cada uma consiste em colunas e uma chave primária. A chave primária é o nome de uma coluna, esta tem como conteúdo valores únicos. Para uma tabela que simula uma ou mais chaves secundárias, os valores destas chaves secundárias da família de colunas serão os nomes de outras colunas que representam as chaves secundárias.
- A coluna é a menor unidade para armazenar dados (característica de um modelo orientado a coluna), sendo composta pelos campos: nome, valor e um campo chamado *timestamp* contendo um valor de tempo em que aquele dado foi inserido/atualizado. Uma coluna de uma família de colunas deve conter, pelo menos, em cada um de seus campos uma estrutura semelhante à chave primaria, uma chave única chamada de *row key*.
- As linhas, diferentemente das de um banco tradicional, são como um conjunto de colunas que possuem a mesma chave primária [27]. Outra diferença relacionada às linhas é o espaço alocado pelo mecanismo de armazenamento, em um banco relacional já se aloca o espaço para todas as colunas de uma linha ainda que seu valor seja *NULL*. O mecanismo de armazenamento do Cassandra só aloca espaço para as colunas presentes em cada linha [33], possibilitando um menor esforço computacional ao adicionar ou retirar uma coluna de uma tabela e criando dois tipos de famílias de colunas: as estáticas e as dinâmicas.

Famílias de colunas estáticas são aquelas que usam um conjunto de nomes de colunas mais estático, ou seja, não são criadas novas colunas durante o acesso ao banco. Ela assemelha-se mais a um banco relacional, como pode ser visto na Figura 4.1 (a) os campos vazios são poucos e os nomes das colunas são os mesmos para cada *row key*. As famílias de colunas dinâmicas fazem um uso diferente para cada célula, os dados, em vez de serem armazenados no campo "valor" de uma coluna, são armazenados no campo "nome" da coluna. Essa estratégia é usada para se ter uma maior eficiência em consultas, pois os dados podem ser pré computados e então armazenados nos campos dos valores e cada linha funciona como uma *view* [19]. A Figura 4.1 (b) mostra um exemplo de famílias de colunas dinâmicas que fazem uso dessa estratégia descrita.



(a) Exemplo de uma família de colunas estática



(b) Exemplo de uma família de colunas dinâmica

Figura 4.1: Exemplo de uma família de colunas estática e uma dinâmica. Adaptado de [19].

Famílias de colunas dinâmicas só existem por duas características do Cassandra, a primeira, já discutida, é a possibilidade de criar uma família de colunas com um número variável de colunas sem prejudicar a alocação de espaço requisitada pelo banco. A segunda característica é o fato da possibilidade de inserir dados em uma coluna sem definir antecipadamente quais tipos de dados serão recebidos pelos campos "valor". Essa característica é chamada de *schema-free* ou *schemaless* [40, 33] e somente por isso podem ser trabalhados os dados que estão no campo nome e inseridos no campo valor sem ser definido seu tipo. No entanto apesar de ainda ser possível fazer uso de uma família de colunas

schema-free, foi considerado que para o desenvolvedor de um banco a definição de tipos facilitaria este trabalho, portanto, a partir da versão 0.7, o Cassandra tem desencorajado o uso dessa ferramenta e pode ser considerado como *schema-optional* uma vez que não é obrigatório seu uso [40].

4.1.3 Interação com o Banco

As requisições de escrita no banco são interpretadas de forma diferente do banco relacional. Quando uma escrita ocorre, o Cassandra armazena os dados em uma estrutura na memória RAM chamada *memtable*, já a instrução de escrita é gravada em uma estrutura no disco, o *Commitlog*, como pode ser visto na Figura 4.2. Estas escritas no disco (*Commitlogs*) permanecem ainda que no caso de uma falha de hardware [22]. Ao serem executadas as escritas no banco, o Cassandra aloca dinamicamente mais memória para a *memtable*, somente quando ultrapassado certo limite de memória que estes dados são descarregados em disco nas SSTables, e só então o *Commitlog* é apagado [22]. Portanto, para um dado ser escrito de forma mais rápida no disco rígido é preciso restringir a quantidade de memória limite para a *memtable*, pois assim os dados serão escritos de forma mais rápida no *Commitlog*, que fica em disco e não na memória RAM, porém isto diminui a performance da consulta.

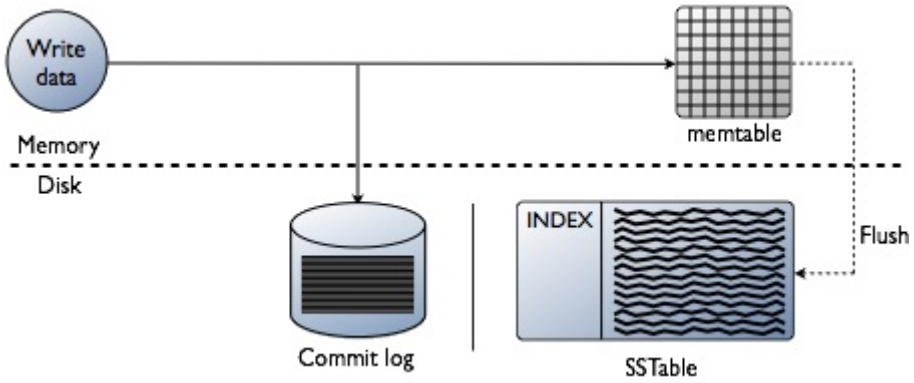


Figura 4.2: Estruturas da escrita e leitura no banco [22].

A instrução de deletar também é diferente, uma tabela não é deletada logo após a instrução. A tabela é marcada na SSTable como uma tabela apagada e um processo chamado de compactação que executa a instrução e também agrupra fragmentos de linhas para otimizar a pesquisa que busque um dado no disco [20].

Como o Cassandra é orientado a coluna, ao executar uma instrução de leitura de uma linha, é preciso agrupar todas as SSTables que contém colunas presentes na linha pesquisada. Porém, antes de executar tal operação, uma estrutura chamada de *Bloom filter*, presente em cada SSTable, confere se a SSTable possui algum dado presente naquela consulta [21]. Outra estratégia que o Cassandra utiliza para melhorar a pesquisa é o uso de uma memória cache para as *row keys*, que carregam-se todas as *row keys* de determinada tabela para a memória e para a linha que a carrega quando buscada [21].

4.1.4 Limitações

Na versão atual, uma das únicas limitações estruturais que realmente pode ser um problema, quando se usa o Cassandra em condições normais, é o limite do número de células. Em uma partição, a quantidade máxima de células (linhas X colunas) é de 2 milhões [7].

As outras limitações estão relacionadas a linguagem Java, como o limite da área de memória que a máquina virtual pode usar, ou são ligadas as consultas ao Cassandra, como será abordado na Seção 4.2.

Outra característica que não é uma limitação, mas que é um fator que aumenta a dificuldade de trabalho é a instalação complicada do banco, que pode ocasionar diferentes erros para usuários menos cuidadosos.

4.2 Arquitetura do Sistema

Antes de descrever a estrutura do Cassandra propriamente dita, existem alguns conceitos básicos que podem ser vistos para facilitar o seu entendimento. A seguir estão alguns deles [26]:

- *Cluster*: Um grupo de nós onde se armazena os dados. Também é possível criar um *cluster* de nó único.
- Nó: Uma instância física do Cassandra. Como visto, não há diferença entre um nó e outro, pois o banco é descentralizado.
- Replicação: É o processo de armazenamento de cópias dos dados em vários nós para garantir as características de confiabilidade e tolerância a falhas. O número de cópias é definido pelo fator de replicação.
- Particionador: Um particionador serve para distribuir os dados de maneira uniforme entre os nós do *cluster*, para que a carga seja balanceada.
- *Data Center*: é um grupo de nós que estão configurados em conjunto dentro de um mesmo *cluster* para fins de replicação. Não é necessariamente um *data center* físico.
- CQL: Abreviação de *Cassandra Query Language* é uma linguagem de *script* usado para ser a interface com o usuário do banco. A abstração de uma tabela possuindo linhas e colunas é bastante semelhante com a da linguagem SQL usada em bancos relacionais. Uma diferença entre os dois é que o CQL é mais simplificado e não suporta alguns recursos como *joins*, presentes em bancos que utilizam SQL.

O Cassandra é projetado para lidar com uma grande quantidade de dados em vários nós, buscando eliminar a possibilidade de erros. Sua arquitetura é baseada no entendimento de que falhas de sistema e de hardware podem e devem, acontecer. Ele aborda essas condições de falhas através do emprego de um sistema distribuído onde todos os nós estão na mesma posição hierárquica e os dados são distribuídos entre todos os nós do *cluster*.

Todos os nós constantemente trocam informações pelo *cluster* e simultaneamente são gerados relatórios em cada nó contendo informações de todas as escritas para se manter a

durabilidade dos dados. Os dados também são gravados em uma estrutura na memória, chamado de *memtable* e escrita em um arquivo chamado de *SSTable* em disco quando a estrutura em memória estiver cheia. Todas os dados gravados são automaticamente particionados e replicados em todo o *cluster* [23].

A arquitetura do Cassandra permite que um usuário (autenticado por *login* e senha) se conecte a qualquer nó em algum *data center* e acesse os dados utilizando a linguagem CQL. Normalmente, um *cluster* apresenta uma *keyspace* por aplicação. Os desenvolvedores podem executar comandos CQL através do programa *cqlsh* ou através de *drivers* em diferentes linguagens de programação [23].

4.2.1 Distribuição e Replicação de Dados

A distribuição e replicação de dados estão relacionadas. Isso acontece porque ele é concebido como um sistema rodando em uma rede sem hierarquia entre nós, esse sistema faz cópias dos dados e distribui as cópias entre um grupo de nós. Os dados são organizados por tabela e identificados com uma chave primária. Essa chave primária determina em qual nó os dados serão escritos, cópia das linhas também são escritas e chamadas de réplicas [23].

Nós Virtuais

Nós virtuais ou *vnodes* são uma mudança de paradigma. Antes cada nó possuia apenas um único espaço de memória em disco para armazenar os dados do Cassandra e cada nó possuia um único *token*, um número inteiro que representava o início da memória disponível. Os *tokens* são definidos e distribuidos entre os nós pelo particionador de tal forma que os *tokens* são únicos, crescentes e que o valor do *token* subsequente determina o fim do espaço de memória do anterior e o início do espaço de memória em um próximo nó. Com a criação dos *vnodes* um nó pode ter mais de um *token*. A quantidade de nós virtuais dentro de um mesmo nó físico (estabelecida pelo desenvolvedor) que dita quantos *tokens* um nó físico terá. Estes *vnodes* são uma maneira de se simular um número maior de nós dentro de um nó real, inclusive são tratados pelo particionador como um nó real e por isso trazem algumas vantagens, são elas [23, 28]:

- Otimizar o armazenamento quando cada linha a ser armazenada tem um tamanho pequeno.
- Não é necessário usar métodos para reequilibrar um *cluster* ao se adicionar ou remover nós. Quando um nó se junta ao *cluster*, ele assume a responsabilidade por uma parcela de dados de outros nós. Se um nó falhar, o seu conteúdo é distribuído uniformemente entre outros nós do *cluster*.
- A reconstrução de um nó sem conexão é mais rápida, pois envolve todos os outros nós do *cluster*.
- Facilita o uso de máquinas diferentes em um *cluster*. É possível colocar um número proporcional de *vnodes* em máquinas com capacidades de armazenamento diferentes entre si.

A distribuição dos nós dentro de um *cluster* pode ser mais facilmente entendida se for feita uma analogia da forma de um anel. Cada nó é responsável por armazenar dados cuja chave primária está dentro de uma faixa de valor. A Figura 4.3 ilustra essa estrutura de anel e mostra como se comportam os *vnodes* dentro dela, como pode ser observado na figura os *vnodes* são criados sequencialmente de tal forma que não alteram a estrutura do anel.

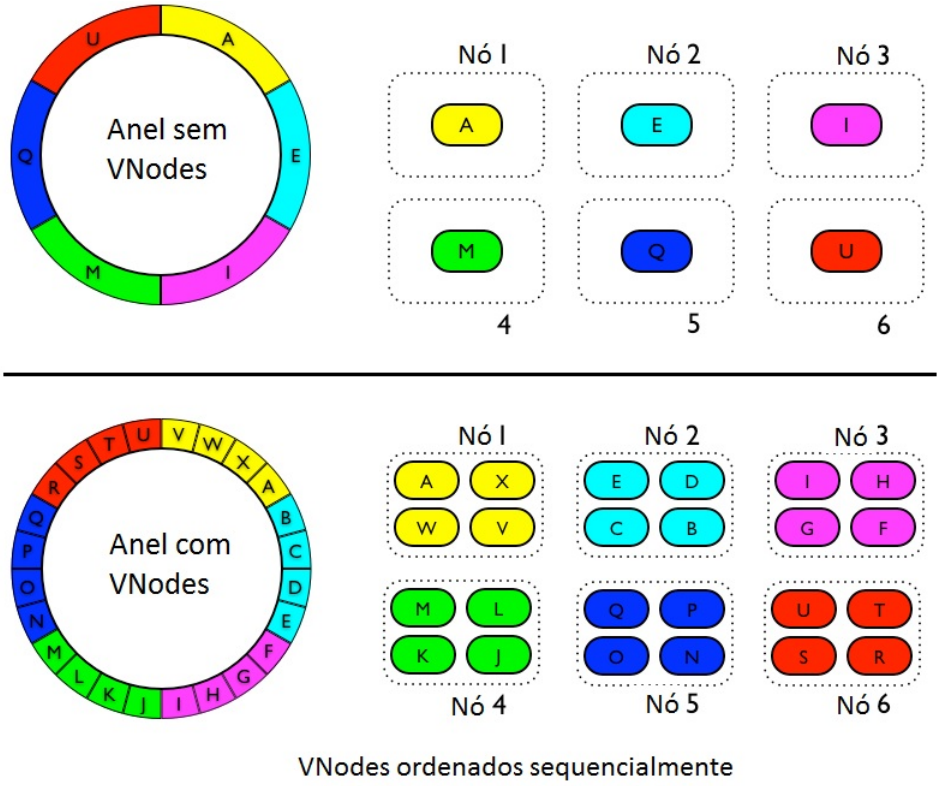


Figura 4.3: Estrutura de um *cluster*. Adaptado de [29]

Particionador

Um particionador é um gerador e distribuidor de *tokens*. O *token* é um elemento essencial para a arquitetura do anel, por isso é chamado de *partitioner-dependent*. Cada nó tem um único e distinto *token*, pois esse *token* determina a extensão do armazenamento dos dados de cada nó. Essa extensão que abrange um *token* a outro é uma abstração das chaves de uma linha, *row keys*, atreladas a cada dado inserido no banco, que serão armazenadas naquele nó. Assim a extensão de um nó inicia-se no *token* do nó atual até o próximo valor de *token* no anel. Para gerar e distribuir esses *tokens*, o Cassandra oferece três opções de particionadores para que o mais adequado seja escolhido para a aplicação, os principais particionadores são [18]:

- O *RandomPartitioner* distribui os dados de forma que estejam balanceados pelo anel. Tal distribuição do dado é feita a utilizando um *hash* MD5 sobre o valor de cada *row key*. Assim os *tokens* gerados são do tipo inteiro com valores de 0 a 2127-1.

- *ByteOrderedPartitioner* é geralmente usado para uma partição de forma ordenada. As *row keys* não passam por uma função de *hash*, elas são ordenadas lexicamente pelo *byte* da *row key*. Assim os *tokens* gerados são do tipo *string* com valores de (espaço em branco) até ∞. Esse particionador tem como desvantagem o caso das *row keys* serem muito similares, fará com que os *tokens* tenham valores próximos e assim estes dados serão armazenados em um mesmo nó, fazendo com que o anel seja desbalanceado e criando um gargalo na consulta, pois não se estará utilizando o potencial das outras máquinas no anel.
- O *Murmur3Partitioner* provê uma performance melhor que o *RandomPartitioner* e tão aleatório quanto, pois cria um *hash* de 64-bits(com valores entre -2^{63} e $+2^{63}$) da *row key*.

A Figura 4.4 ilustra um *cluster* com *vnodes* fazendo uso de um particionador aleatório. A vantagem em relação da estrutura da Figura 4.4 em relação à Figura 4.3 (b) é que os dados ficam mais dispersos e o anel melhor balanceado. Por exemplo, quando são inseridas 4 linhas em um banco o Casandra inserirá cada uma delas em um *vnode* diferente, assim as 4 linhas serão distribuídas nos *vnodes*: A, B, C e D. Conforme a Figura 4.3 (b) vemos que estas linhas estarão apenas no nó físico 1 e 2, enquanto que em um anel que tem seus nós vituais distribuídos aleatoriamente, representado na Figura 4.4, estas mesmas linhas estão nos nós físicos: 2, 3, 5 e 6. A vantagem de se ter um anel bem distribuído é que a consulta é melhorada uma vez que é possível fazer mais consultas em paralelo.

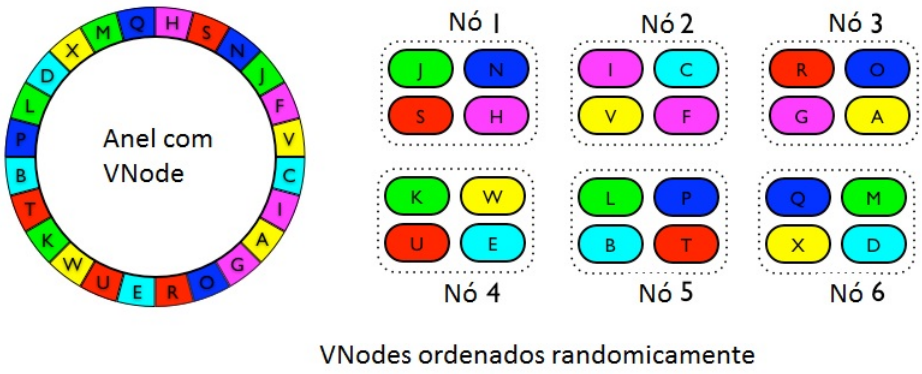


Figura 4.4: Estrutura de um *cluster* com particionador aleatório. Adaptado de [29]

4.2.2 Níveis de Consistência

Como visto anteriormente, consistência refere-se a forma como são mantidas as linhas atualizadas e sincronizadas em todas as suas cópias, tendo como objetivo de que sempre que for feita uma leitura, o dado lido estará na sua versão mais recente.

Analisando essa situação, o Cassandra apresenta o conceito de consistência ajustável (*tunable consistency*) em que para cada leitura ou escrita, a aplicação cliente pode decidir quão consistentes os dados solicitados devem estar. Isso é feito através de alterações no caminho que os dados fazem e no número de copias geradas. Além disso, o Cassandra também apresenta uma série de mecanismos embutidos para assegurar que os dados permaneçam corretos em suas diferentes réplicas [23].

Consistência de Escrita

Os níveis de consistência podem ser ajustados para conseguir o equilíbrio entre tempo de resposta e acurácia. No caso da leitura, seguem alguns dos níveis e suas características [23]:

- *ANY*: Fornece baixa latência e a garantia de que uma escrita nunca irá falhar. Oferece a menor consistência e a maior disponibilidade em relação a outros níveis.
- *ONE*: Usado pelo maior número de usuários, pois os requisitos para conseguir a consistência não são rigorosos. O nó de réplica mais próximo do nó que recebeu o pedido é responsável por atender o pedido, a não ser que o sistema identifique que aquele nó não está com um desempenho aceitável, nesse caso o controle é desviado para outro nó.
- *QUORUM*: Fornece uma consistência forte mas com a possibilidade de falha.
- *ALL*: É o que possui a maior consistência e a menor disponibilidade entre todos os níveis.

De modo geral, em todos os níveis são geradas cópias para as réplicas dos nós. Mesmo as que estão em outra central de dados. A principal diferença entre os níveis é o número de réplicas que se exige uma resposta informando que recebeu os dados.

Consistência de Leitura

Semelhantemente, existem os níveis de consistência de leitura. O Cassandra verifica o número de réplicas que enviaram os dados e quão recentes esses dados são, baseado na variável de tempo contida em cada linha. Seguem alguns níveis [23]:

- *ONE*: Fornece a mais alta disponibilidade de todos os níveis, porém apresenta a maior probabilidade de serem lidos dados antigos. As réplicas lidas podem não possuir a versão mais recente dos dados.
- *QUORUM*: Garante uma forte consistência se for tolerável um certo nível de falha.
- *ALL*: Fornece a mais alta consistência de todos os níveis junto com a menor a disponibilidade também.

Capítulo 5

Estudo de Caso e Implementação

Neste capítulo são apresentados dois estudos de caso em que os dados da bioinformática são inseridos e retirados de um banco do Cassandra, um contendo dois nós, outro contendo quarto nós. Além diso é feita uma comparação do desempenho desses mesmos testes em um modelo relacional [41]. Na Seção 5.1 são apresentadas as informações referentes aos arquivos de entrada. A Seção 5.2 descreve o ambiente de nuvem criado e também como foi feita a elaboração do banco.

5.1 Características dos Dados da Bioinformática

Como visto antes, o objetivo desse trabalho é a obtenção de resultados em testes de desempenho inserindo dados da bioinformática no Cassandra. Os dados a serem inseridos são arquivos biológicos no formato FASTQ [52]. O formato FASTQ trata-se de um arquivo de texto que serve para armazenar uma sequência biológica (geralmente sequência de nucleotídeos) e seus índices de qualidade correspondentes. Os dados são codificados usando caracteres ASCII para serem abreviados [52]. A Figura 5.1 mostra um trecho de um arquivo FASTQ como exemplo.

```
@SRR002323.4 080317_CM-KID-LIV-2-REPEAT_0003:3:1:120:333 length=36
TAATCAGTAAAGAATTGATGTCCCTATTTGGTGAAT
+SRR002323.4 080317_CM-KID-LIV-2-REPEAT_0003:3:1:120:333 length=36
IIIIIIIIIIII=&IIIIIIIIIIIIIIIII%II
@SRR002323.5 080317_CM-KID-LIV-2-REPEAT_0003:3:1:115:432 length=36
TGTAGAGTGGACATGCTTAGATCCAGCAACTGATAT
+SRR002323.5 080317_CM-KID-LIV-2-REPEAT_0003:3:1:115:432 length=36
IIIIIIII:IIIIIIIIIIIIIII/IIIIIIII
```

Figura 5.1: Exemplo de FASTQ

O arquivo é dividido em blocos, chamados de SRSs (*short read sequence*), de 4 linhas que seguem uma periodicidade.

Os seis arquivos utilizados para os estudos de caso foram usados em uma dissertação de mestrado [41], dos quais três são resultantes da filtragem dos dados resultantes do

sequenciamento de amostras de células do rim e os outros três de células do fígado. No capítulo 6 são mostrados mais dados desses arquivos. O sequenciamento de amostras de células é um processo bioquímico que tem como objetivo determinar a ordem das bases nitrogenadas da molécula de DNA e a filtragem é o processo que tem por objetivo retirar as SRSs que tem uma baixa qualidade e podem afetar negativamente os estudos e trabalhos com estes arquivos [41].

Para esse trabalho não se faz necessário o conhecimento preciso do processo de obtenção desses dados biológicos, uma vez que o foco é a performance do armazenamento e retirada deles no banco de dados. Não são necessárias alterações em seu conteúdo, apenas a garantia de que não serão perdidas informações ao longo das inserções e retiradas.

5.2 Desenvolvimento da Aplicação Cliente

Nessa seção é detalhado como foi preparado o ambiente onde ocorrem os testes. Mostrando inclusive as ferramentas que foram descartadas ao longo desse processo.

Pentaho

O Cassandra é um software livre, sendo assim são desenvolvidas diversas distribuições de terceiros para criar uma interface mais amigável, incluir mais recursos ou softwares de integração. Dentre essas soluções foi-se considerado o Pentaho [13], um software de *Business Intelligence* (BI) para integração de Big Data. Apesar de sua interface intuitiva, foi verificado após testes que o Pentaho é um software muito restrito. Não é possível gerenciar os nós ou a maneira como é feita a distribuição dos dados. Ele também apresentou incompatibilidades com a linguagem CQL e um poder muito pequeno de configurações. Como esses pontos levantados limitam o uso dos recursos que otimizam as interações com o banco, o Pentaho foi desconsiderado para este projeto.

Cassandra-cli

Todas as distribuições do Cassandra trazem consigo dois clientes simples com interface pelo *shell* para fazer iterações com o banco, o *Cassandra-cli* e o *cqlsh*. O *Cassandra-cli* é uma ferramenta muito útil, sua linguagem de consulta é o CQL versão 2 e conta com muitos recursos não existentes no Pentaho, dentre eles, que um *script* seja executado juntamente a sua inicialização para fazer iterações no banco. Foi considerada a possibilidade de tratar o arquivo FASTQ inicial para criar um *script* a ser executado prontamente com o *Cassandra-cli*. Apesar de possuir essa funcionalidade bastante útil, o *Cassandra-cli* não tem suporte para criar nem trabalhar com famílias de colunas utilizando o CQL versão 3 [16]. Isso é um problema, pois o CQL versão 3 é o primeiro a utilizar *virtual nodes*, que aceleram o processamento como visto no capítulo anterior, além de ser recomendado pela Apache a descontinuidade do uso do *Cassandra-cli* [5]. Sendo assim, essa abordagem também foi desconsiderada.

DSE

Por fim foi utilizado o *DataStax Enterprise Edition* (DSE) versão 3.1, que é uma distribuição de terceiros (*Third Party Distribution*) do Cassandra fornecida pela empresa

DataStax. Segundo a empresa Datastax, o DSE é uma plataforma para dados *Big Data* construída tendo como base o Apache Cassandra que gerencia e analisa dados em tempo real, feita para maximizar o processamento dos bancos Cassandra, Apache Hadoop e de uma ferramenta de busca chamada Apache Solr [6]. Diferentemente das ferramentas anteriores que eram apenas clientes que se conectavam ao banco, o DSE é uma plataforma completa, sendo possível até mesmo gerenciar a instalação em múltiplos nós. Assim a estratégia final foi a criação de um cliente próprio que se conecta ao banco e mantém as operações com a ferramenta gerenciadora de *clusters* via *browser* chamada OpsCenter, que faz parte do DSE.

Características da Aplicação

Assim ao elaborar a aplicação cliente estabeleceu-se como requisitos funcionais que o sistema:

- crie um *keyspace*;
- crie uma tabela que armazenará um arquivo FASTQ;
- crie uma tabela com o nome dos FASTQs inseridos e os seus metadados;
- receba um arquivo de entrada contendo os dados do arquivo FASTQ e os insira em uma tabela já criada, assim como seus metadados e nome do arquivo em outra tabela;
- extraia todo o conteúdo de uma tabela com o conteúdo de um arquivo FASTQ;
- deleta a tabela e o *keyspace*.

Como requisito não funcional busca-se:

- Utilizar-se de uma API Java fornecida pela DataStax para se ter uma maior integração entre a distribuição do Cassandra e a aplicação cliente desenvolvida;
- Uso da linguagem CQL3 para as iterações com o banco, pois esta é a linguagem de consulta atual do Cassandra e assemelha-se com o SQL;
- Uso de arquivos JSON pelo cliente, que faz a inserção e retirada do banco. Essa estratégia foi adotada por ser mais simples tratar arquivos JSON em Java;
- Alto desempenho nas operações que fazem muito acesso ao banco;
- Pouco espaço ocupado pelo banco nos servidores;
- Consistência dos dados extraídos do banco, esse é o principal requisito a ser buscado, pois caso não sejam consistentes o mapemento genético não terá os resultados corretos.

Primeira Aplicação

Foram criadas três aplicações (ou programas), a primeira chamada de "fastqTojson" serve para fazer o tratamento no arquivo FASTQ de entrada e dividí-lo em arquivos JSON menores. Após a execução, cada arquivo JSON gerado possuía 500 mil SRSs, que seriam

inseridas no banco em 10mil linhas, cada linha sendo um agrupamento de 10 colunas e cada campo "valor" de uma coluna contendo 5 SRSs. Esse programa foi implementado em linguagem C, tendo como objetivo a simplicidade da implementação e a busca por uma boa performance nessa atividade secundária. Assim influênciar de maneira pequena o resultado final.

Segunda Aplicação

A segunda aplicação é o cliente do Cassandra desenvolvido em Java utilizando a API da DataStax. Este cliente executa as seguintes operações: criação de um *keyspace*, inserção de todos os arquivos JSON resultantes da primeira aplicação em uma única tabela e a extração de uma tabela completa. Todas as operações primeiramente se conectam ao *cluster* em que o Cassandra da máquina foi configurado, e são desconectadas quando finalizadas.

Algumas configurações do Cassandra só são definidas no momento da criação de seu esquema (*keyspaces* e famílias de colunas). Outras, descritas na seção seguinte, têm de ser definidas antes mesmo de se criar o ambiente de nuvem (criar o *cluster*). Foi criado um único *keyspace* de nome "bioData" para o *cluster*, que foi chamado de "BIOCluster", dentro do *keyspace* haviam todos os arquivos FASTQ. Para se criar um *keyspaces*, são obrigatórios os seguintes campos: nome, a estratégia de alocação das réplicas e o fator de replicação. A criação de *keyspaces* e tabelas, segue o modelo de dados da Figura 5.2.

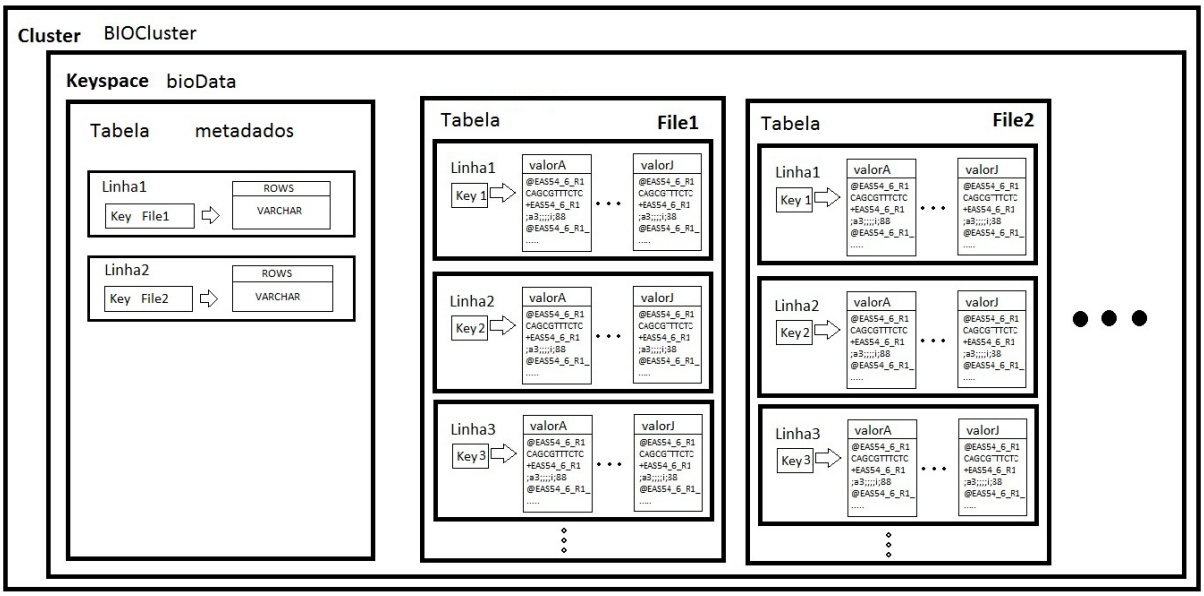


Figura 5.2: Modelo de dados do banco

A estratégia de alocação das réplicas determina se elas serão distribuídas por uma rede de diferentes *cluster*, no caso a "Estatégia de Topologia em Rede", ou se serão distribuídas em um único *cluster*, no caso da "Estatégia Simples" [6]. Foi escolhida a Estratégia Simples, por ser mais adequada a um ambiente de testes estável. O fator de replicação, abordado no capítulo passado, trata de quantas réplicas serão distribuídas ao longo do anel. Por tratar-se de um ambiente de testes, em que existia um controle para que não houvesse falhas em nenhum nó e pelo objetivo da monografia ser voltado à performance e

consistência, o número de réplicas ao longo do anel é de apenas uma. Tendo que o número de réplicas interferem no tempo.

Após criado o *keyspace* é possível executar a operação de inserção no banco. Antes da inserção, é criada uma tabela de mesmo nome que o arquivo FASTQ original com onze colunas, das quais dez abrigam uma pequena parte de um arquivo JSON (cerca de 10 SRRs), cada uma delas tem aproximadamente 1MB de tamanho. Um conjunto pequeno de colunas foi escolhido, pois o Cassandra utiliza o particionador determinado pelo *cluster* para armazenar cada linha em um nó ou nó virtual diferente para, dessa forma, usar ao máximo os recursos de todos os nós ao se executar uma leitura [28]. Ao criar uma tabela em que as linhas são um conjunto de muitas colunas, a performance da leitura é prejudicada, da mesma forma a escolha de um particionador que armazena cada linha em um nó ou em um conjunto pequeno de nós. Assim que criada a tabela, todos os arquivos JSON resultantes da primeira fase são lidos sequencialmente e inseridos nesta tabela. Ao fim da inserção uma única linha é inserida na tabela de metadados contendo o número de linhas que aquela tabela possui e sua *row key* é o próprio nome de arquivo FASTQ.

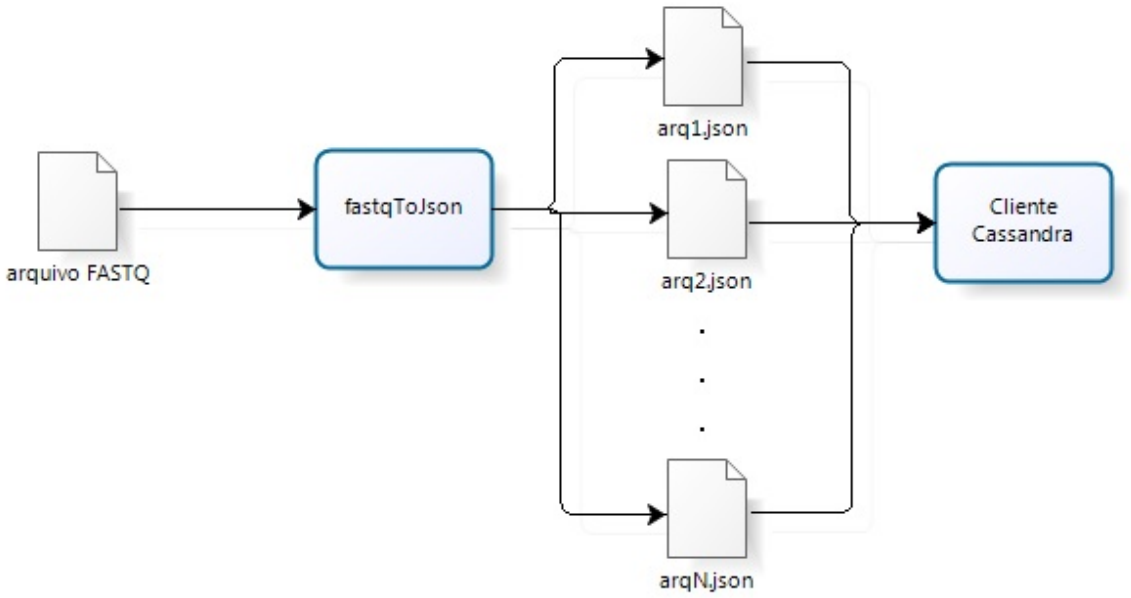


Figura 5.3: Etapas da Inserção

Para a extração de uma tabela, primeiro é feita uma consulta à tabela de metadados pela chave que contém o nome do FASTQ a ser retirado e o número de linhas a ser retirado é guardado. Por ser uma aplicação Java e os arquivos inseridos nas tabelas serem muito grandes esta foi a melhor solução para que a memória da *Java Virtual Machine* não ultrapassasse o limite, nem fosse necessário percorrer toda a tabela contendo o FASTQ para encontrar o maior valor de linha. É importante enfatizar que o CQL não possui nenhum comando que ordene a tabela, pois como o Cassandra é voltado a Big Data, ordenar uma tabela torna-se uma tarefa que consumiria quase todos os recursos, pois teria que buscar tais valores em todos os nós do anel. Assim que o número de linhas é

obtido a seleção é feita linha a linha e escrita em um arquivo. Este arquivo é temporário, a ser tratado pela próxima aplicação.

Terceira Aplicação

A terceira aplicação transforma o arquivo temporário de saída no FASTQ definitivo, uma cópia do FASTQ original. Ela apenas faz alterações no arquivo para que fique idêntico original de entrada, também foi feita em linguagem C. Para a execução de testes foi criado um *script* para execução no *shell* que automatiza o processo e apaga, a cada fase, os arquivos temporários.

Nas Figuras 5.3 e 5.4 são mostadas as aplicações e a maneira como elas se relacionam nos processos de inserção e retirada respectivamente.

Figura 5.4: Etapas da Extração

5.3 Arquitetura do Ambiente de Nuvem

Para criar um ambiente distribuído, é necessário instalar em cada máquina (que irá compartilhar os recursos) um Cassandra-agent. Para estabelecer uma conexão entre todos os agentes, é necessária a configuração de três arquivos ("cassandra.yaml","cassandra-topology.properties" e "opscenterd.conf") presentes em cada agente após a instalação do Cassandra. É importante ressaltar que a edição desses três arquivos possibilita o controle completo de todas as ferramentas à disposição do Cassandra. Porém nesse trabalho foram utilizadas algumas representações gráficas dos dados presentes, obtidas por meio do software OpsCenter [30] (software da empresa DataStax) que dispõe de uma interface via *browser* para configuração, gerência e análise do ambiente de nuvem.

Então, pelo OpsCenter, foi criado o *cluster*, durante sua criação foram configurados: o particionador e os *vnodes*. Uma vez feita a escolha do particionador não é possível mudá-la sem antes apagar todo o conteúdo da tabela. Foi selecionado o MurMur3Partitioner como particionador, pois, como citado no Capítulo 4, é o que melhor distribui os dados de forma balanceada ao longo do anel. Como citado no Capítulo 4, uma maior quantidade de nós virtuais é mais adequado quando o tamanho de cada linha é pequeno. Nesse estudo de caso cada linha tem apenas dez colunas, um tamanho em torno de 1MB, assim foi selecionado para que cada nós possuísse 256 nós virtuais pra otimizar o armazenamento, pois 256 é o maior valor recomendado para nós virtuais dentro de um nó físico [6].

Capítulo 6

Resultados e Discussão

Neste capítulo são apresentados os resultados dos dois estudos de caso, um contendo dois nós, outro contendo quarto nós. A Seção 6.1 descreve os resultados da inserção e extração dos dados e uma discussão sobre o ganho de desempenho com o aumento do número de máquinas e a seção 6.2 apresenta uma comparação dos resultados do estudo de caso feito com os resultados de um estudo de caso semelhante realizado em um banco de dados relacional.

Para a avaliação da performance e consistência dos dados no banco foram realizados dois estudos de caso: um com uma nuvem constituída por duas máquinas, outro uma nuvem constituída por quatro máquinas. O aumento no número de máquinas tinha por objetivo avaliar a escalabilidade do Cassandra e avaliar a melhora da performance do banco. A Nuvem criada com duas máquinas consistiam de uma máquina HP Proliant Ml110G7 com processador Intel Xeon E3-1220/3.1 GHz com 6GB de memória RAM e outra máquina semelhante, mas com 8GB de memória RAM. Para uma segunda etapa de testes, foram adicionadas mais duas máquinas ambas com processador Intel Core i7 e 4GB de memória RAM. Nos dois casos, cada máquina utilizava o sistema operacional Ubuntu versão 12.04, possuía um IP diferente.

Os dados usados na nuvem computacional com duas e quatro máquinas são os mesmos, os seis arquivos FASTQ descritos anteriormente. Estes mesmos dados foram utilizados em uma dissertação de mestrado da UnB que tratava da inserção e extração em um banco relacional. Na dissertação de mestrado foi utilizado apenas uma máquina, um servidor, HP (8 Intel(R) Xeron(R) de 8 CPU's de 2.13GHz e 32GB de memória RAM sobre o sistema operacional Linux Server Ubuntu/Linaro 4.4.4-14 [41]. Uma vez que os dados da dissertação de mestrado e desta monografia são os mesmos é possível comparar com um banco relacional e um NoSQL. Apesar da configuração da máquina utilizada no caso do banco relacional ser superior à soma de todas as configurações das quatro máquinas usadas no banco do Cassandra, foram utilizados os resultados do banco relacional para demonstrar que: com a escalabilidade da computação em nuvem é possível atingir uma alta performance em um paradigma de bancos de dados sem o uso de um servidor.

Primeiramente para o estudo de caso em um banco NoSQL foram feitos testes de criação de *keyspaces*, tabelas, inserção e retirada de dados FASTQ a nível de *localhost* e trouxeram resultados positivos, mostrando-se possível a execução dessas operações com o cliente criado.

Feito isso, foi montada a estrutura proposta, composta inicialmente por dois nós e executando o Cassandra na distribuição DSE. A Figura 6.1 mostra a interface do OpsCenter, onde se pode ver o modelo de anel contendo os dois nós, o espaço ocupado pelos dados já inseridos é mostrado no campo *Total Size* com o valor de 5.6 GB. Na figura também pode-se ver o menu lateral com as opções de trocar entre as diferentes visões do banco e configurações.

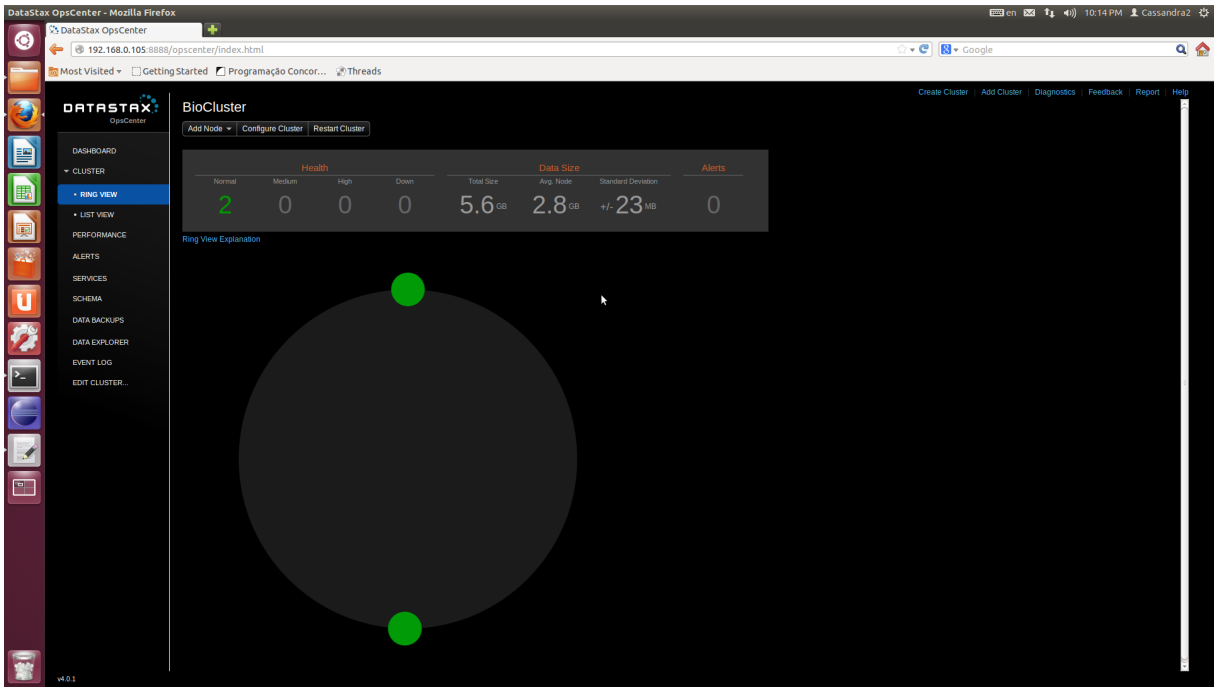


Figura 6.1: Interface do OpsCenter

6.1 Inserção e Extração dos FASTQ Referentes ao Fígado e Rim

Como entrada, inicialmente, foram usados três arquivos FASTQ com dados filtrados tirados de células do fígado. Na Tabela 6.1 são apresentadas as seguintes colunas: nome dos arquivos, tamanho arredondado e real e o número de linhas que o arquivo JSON possuia quando foi inserido no banco.

Tabela 6.1: Arquivos Fígado

Nome	Tamanho	Número de Linhas
SRR002321	9,0 GB (9.046.495.358 bytes)	850.933
SRR002322	4,0 GB (4.016.123.172 bytes)	358.841
SRR002323	3,2 GB (3.202.734.904 bytes)	286.563

Em seguida, foram inseridos arquivos contendo os dados das células do rim. Tais arquivos usados são mostrados na Tabela 6.2.

Tabela 6.2: Arquivos Rim

Nome	**Tamanho**	**Número de Linhas**
SRR002320	6,9 GB (6.891.651.350 bytes)	648.612
SRR002324	3,8 GB (3.757.860.934 bytes)	335.937
SRR002325	5,3 GB (5.328.126.424 bytes)	475.210

Primeiramente foi testado o desempenho das inserções e retiradas desses arquivos usando uma rede composta por duas máquinas com o Cassandra instalado. A Tabela 6.3 mostra os resultados aproximados de inserção e extração para cada um dos seis arquivos. O campo tamanho foi mantido nessa figura para facilitar a visualização da sua relação com o tempo.

Tabela 6.3: Duas Máquinas

Nome	**Tamanho**	**Tempo Inserção**	**Tempo Extração**
SRR002321	9,0 GB	14m30s645ms	23m37s964ms
SRR002322	4,0 GB	6m10s471ms	9m41s018ms
SRR002323	3,2 GB	5m05s914ms	7m39s188ms
SRR002320	6,9 GB	11m25s899ms	14m25s120ms
SRR002324	3,8 GB	6m09s417ms	8m37s890ms
SRR002325	5,3 GB	8m43s330ms	12m23s855ms

O teste seguinte foi o da inserção usando uma rede composta por quatro máquinas, tendo como objetivo verificar se o aumento do número de *clusters* iria alterar o desempenho do banco de dados. A Tabela 6.4 apresenta resultados para esse teste.

Tabela 6.4: Quatro Máquinas

Nome	**Tamanho**	**Tempo Inserção**	**Tempo Extração**
SRR002321	9,0 GB	11m44s105ms	15m04s158ms
SRR002322	4,0 GB	5m05s710ms	7m34s523ms
SRR002323	3,2 GB	4m51s823ms	6m02s648ms
SRR002320	6,9 GB	8m27s630ms	10m00s031ms
SRR002324	3,8 GB	4m42s386ms	6m05s487ms
SRR002325	5,3 GB	8m05s215ms	9m03s041ms

Pode-se perceber que tanto o tempo de inserção quanto o tempo de extração confirmam a hipótese de que o desempenho do banco melhora quando se usa mais máquinas. Tal

característica está relacionada aos próprios objetivos do Cassandra no que diz respeito à escalabilidade e a política de replicações entre as máquinas a fim de se garantir a velocidade.

Com objetivo de ilustrar melhor a diferença de tempo entre as inserções com duas e quatro máquinas foi criado o gráfico ilustrativo da Figura 6.2.

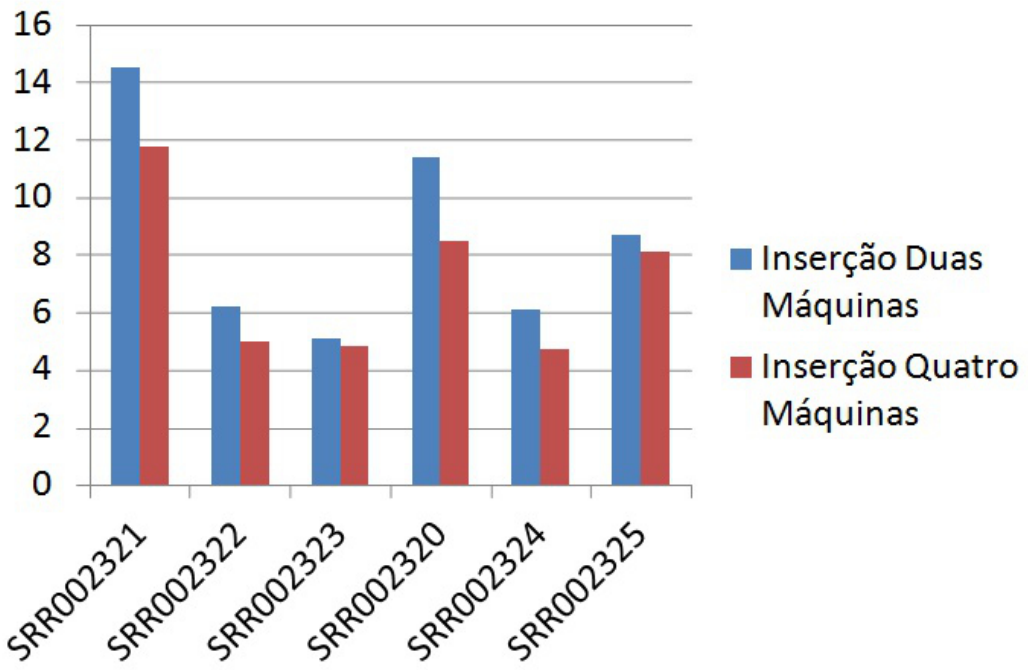


Figura 6.2: Comparativo Entre Inserções

Da mesma maneira, a Figura 6.3 ilustra a diferença entre as extrações com duas e quatro máquinas.

6.2 Comparativo com Banco de Dados Relacional

Finalizando, o teste principal que pôde ser feito nesse trabalho foi uma comparação entre o tempo de inserção e exportação dos dados filtrados da bioinformática usando o Cassandra e usando um banco relacional. Para isso é feita uma comparação entre a soma do tempo de todas as inserções em sequência do rim e do fígado para duas e quatro máquinas, com o resultado medido em outro trabalho [41] que fez o mesmo mas usando uma abordagem relacional. A Tabela 6.5 mostra essa comparação.

Pode-se perceber que o tempo de inserção do Cassandra foi bem menor, devido ao paralelismo e as outras vantagens que a abordagem não relacional apresenta. Entretanto, quando se vai fazer um busca em toda a base de dados, o desempenho dele se mostrou inferior. Porém, não se pode tomar essa medida como um resultado definitivo uma vez que a máquina da implementação relacional era superior ao somatório das características de todas as máquinas usando o Cassandra. Como visto na Seção 4.1.3, a leitura de um banco no Cassandra é muito beneficiada com a escalabilidade, pois, ao adicionar mais

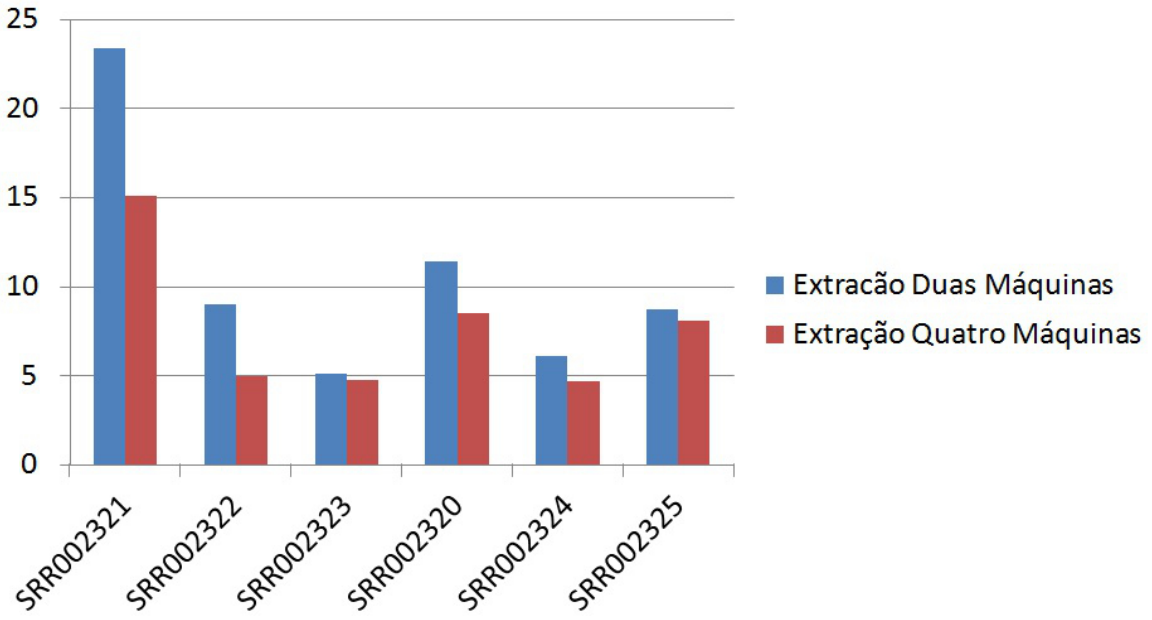


Figura 6.3: Comparativo Entre Extrações

Tabela 6.5: Arquivos Rim

Banco	**Tempo Inserção Total**	**Tempo Retirada Total**
Cassandra (2 máquinas)	52m05s	1h16m25s
Cassandra (4 máquinas)	42m56s	53m49s
Implementação Relacional	1h51m54s	28m27s

máquinas, maior é a quantidade de dados que podem ser armazenados na *memtable*, que é a estrutura de mais rápido acesso no Cassandra, pois o conteúdo permanece na memória RAM. Como a quantidade de dados era superior ao somatório do tamanho da memória RAM das máquinas, parte dos dados foi armazenado na *SSTable* e esta estrutura armazena os dados no disco rígido, tornando mais lento a consulta a eles. Na Figura 6.4 as diferenças entre as implementações é mostrada de maneira visual.

Fazer uma busca massiva e que não pode ter nenhuma perda é uma tarefa complexa em um ambiente de nuvem distribuído pois são necessárias comparações entre as réplicas a fim de se assegurar que nenhum dado foi perdido ou corrompido. Por outro lado, a melhora no desempenho quando se aumentou o número de máquinas é um indício de que para um número maior de máquinas o Cassandra pode superar está implementação relacional, o tornando uma opção viável para a bioinformática.

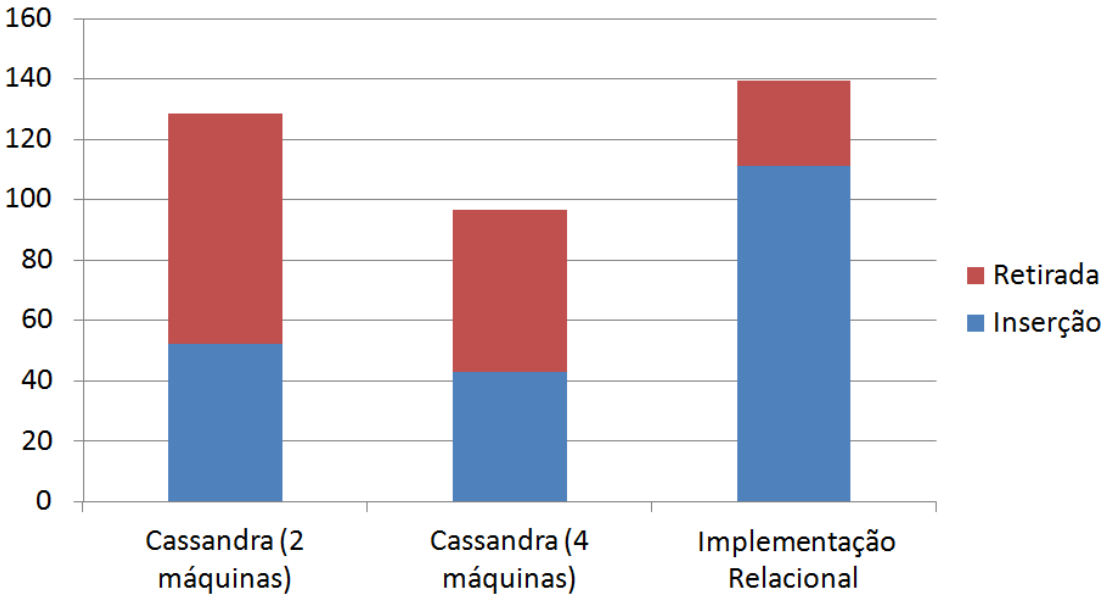


Figura 6.4: Gráfico Comparando Cassandra e Implementação Relacional

Capítulo 7

Conclusões e Trabalhos Futuros

Nessa monografia, foi realizado um estudo acerca da persistência de dados biológicos fazendo uso de uma tecnologia ainda pouco explorada por essa área, que são os bancos de dados em nuvem, especificamente o banco Cassandra. Antes da implementação do estudo de caso, foram analisadas quais as melhores características para desenvolver uma arquitetura de nuvem que possibilitasse a melhor performance e dados consistentes com os originais. Foram avaliadas algumas ferramentas para serem usadas juntamente com o Cassandra, porém foi escolhida a da DataStax Enterprise, uma distribuição do Cassandra em conjunto com o OpsCenter e a API em Java, dentre outras. Tais ferramentas possibilitaram a criação de uma nuvem e uma aplicação cliente que atendia os requisitos ditados pelo modelo de dados e pelos dados biológicos utilizados.

Os resultados obtidos possibilitaram visualizar uma otimização da inserção e consulta ao Cassandra ao se adicionarem mais nós. A inserção teve um ganho de performance de aproximadamente 17% e a retirada dos dados um ganho de mais de 25% se comparados os resultados de duas e quatro máquinas. Em comparação com um banco de dados relacional a performance do banco também foi satisfatória, uma vez que os resultados do uso de quatro máquinas comuns se aproximou do resultado de um servidor contendo o banco relacional. Isso indica uma possível vantagem do Cassandra, uma vez que os recursos computacionais deste banco podem ser cada vez mais escaláveis o que traz um ganho de performance.

Como visto em capítulos anteriores, os bancos não relacionais seguem o teorema CAP, sendo que o Cassandra não difere deles, tendo como características a facilidade de particionamento e a consistência na visualização dos dados em todos os clientes que acessam o banco. Essas duas características permitem que o Cassandra seja um banco que não esteja sempre disponível, pois ao se trabalhar *offline* diferentes visualizações do banco seram criadas e o fundamento da consistência dos dados seria desfeito.

Acredita-se que a proposta aqui apresentada pode trazer um direcionamento no estudo de novas abordagens de persistência de dados *Big Data* biológicos. Os resultados positivos servem de estímulo para o avanço destes estudos, que podem levar à construção de uma aplicação semelhante usando outros bancos NoSQL, variações no número de máquinas em um banco do Cassandra e até mesmo a criação do banco relacional no mesmo ambiente que este estudo de caso foi estabelecido, para uma comparação mais precisa.

Referências

[1] D. J. Abadi. Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12, 2009. 4, 5

[2] D. Abramson, Giddy, and L. Kotler. High performance parametric modeling with nimrod/g: Killer application for the global grid. In *Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, IPDPS '00, pages 520–528, Washington, DC, USA, 2000. IEEE Computer Society. 7

[3] K. M. Albarrak and E. H. Sibley. Translating relational & object-relational database models into owl models. In *Information Reuse Integration, 2009. IRI '09. IEEE International Conference on*, pages 336–341, 2009. 17

[4] A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the resource savings of utility computing models. *Hewlett Packard Laboratories*, (HPL-2002-339), December 2002. 9

[5] Apache.org. CassandraCli. `http://wiki.apache.org/cassandra/CassandraCli`, 2013. [Online; acessado 10-Novembro-2013]. 34

[6] Apache.org. DataStax Enterprise 3.1 Documentation. `http://www.datastax.com/doc-source/pdf/dse31.pdf`, 2013. [Online; acessado 06-Dezembro-2013]. 35, 36, 38

[7] Apache.org. Limitations. `http://wiki.apache.org/cassandra/CassandraLimitations`, 2013. [Online; acessado 6-Dezembro-2013]. 28

[8] R. W. Brito. Bancos de dados nosql x sgbds relacionais:análise comparativa. Technical report, Universidade de Fortaleza, 2010. 17

[9] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid. In *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, volume 1, pages 283–289 vol.1, May 2000. 7

[10] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5–13, 2008. viii, 6

[11] Planet Cassandra. The Five Minute Interview – NASA. `http://www.datastax.com/dev/blog/the-five-minute-interview-nasa`, 2013. [Online; acessado 3-Dezembro-2013]. 23

[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, jun 2008. 25

[13] Pentaho Corporation. Pentaho Big Data Analytics. http://pentaho.com/product/big-data-analytics, 2014. [Online; acessado 10-Janeiro-2014]. 34

[14] Zoho Creator. Platform as a Service. http://www.zoho.com/creator/paas.html, 2013. [Online; acessado 3-Dezembro-2013]. vii, 5

[15] S. Curtis. Netflix foretells 'House of Cards' success with Cassandra big data engine. http://news.techworld.com/applications/3437514/netflix-foretells-house-of-cards-success-with-cassandra-big-data-engine/, 2013. [Online; acessado 3-Dezembro-2013]. 23

[16] DataStax. What's New in DataStax Enterprise 3.1? A Guide for Developers, Architects and IT Managers . http://www.datastax.com/wp-content/uploads/2013/07/WP-WhatsNewDSE31.pdf, 2013. [Online; acessado 6-Dezembro-2013]. 34

[17] DataStax. About Column Families. http://www.datastax.com/docs/1.0/ddl/column_family, 2013. [Online; acessado 6-Dezembro-2013]. 25

[18] DataStax. About Data Partitioning in Cassandra. http://www.datastax.com/docs/0.8/cluster_architecture/partitioning, 2013. [Online; acessado 6-Dezembro-2013]. 30

[19] DataStax. About Data Partitioning in Cassandra . http://www.datastax.com/docs/1.1/ddl/column_family, 2013. [Online; acessado 6-Dezembro-2013]. vii, 26

[20] DataStax. About deletes. http://www.datastax.com/documentation/cassandra/1.2/webhelp/index.html#cassandra/dml/dml_about_deletes_c.html, 2013. [Online; acessado 9-Dezembro-2013]. 27

[21] DataStax. About reads . http://www.datastax.com/documentation/cassandra/1.2/webhelp/index.html#cassandra/dml/dml_about_reads_c.html#concept_ds_vrp_4qx_zj, 2013. [Online; acessado 9-Dezembro-2013]. 27

[22] DataStax. About writes . http://www.datastax.com/documentation/cassandra/1.2/webhelp/index.html#cassandra/dml/manage_dml_intro_c.html#concept_ds_g2s_y1w_zj, 2013. [Online; acessado 9-Dezembro-2013]. vii, 27

[23] DataStax. Apache Cassandra 1.2 Documentation. http://www.datastax.com/documentation/cassandra/1.2/pdf/cassandra12.pdf, 2013. [Online; acessado 6-Dezembro-2013]. 29, 31, 32

[24] DataStax. Big Data: Beyond the Hype Why Big Data Matters to You. http://www.datastax.com/wp-content/uploads/2011/10/WP-DataStax-BigData.pdf, 2013. [Online; acessado 6-Dezembro-2013]. 1

[25] Datastax. Introduction to Apache Cassandra. http://www.datastax.com/wp-content/uploads/2012/08/WP-IntrotoCassandra.pdf, 2013. [Online; acessado 3-Dezembro-2013]. 23

[26] DataStax. Key concepts. http://www.datastax.com/documentation/gettingstarted/getting_started/gettingStartedKeyConcepts_c.html, 2013. [Online; acessado 3-Dezembro-2013]. 28

[27] DataStax. The data model distilled. http://www.datastax.com/documentation/gettingstarted/getting_started/gettingStarteddataModel_c.html, 2013. [Online; acessado 3-Dezembro-2013]. 25

[28] DataStax. Virtual nodes in Cassandra 1.2. http://www.datastax.com/dev/blog/virtual-nodes-in-cassandra-1-2, 2013. [Online; acessado 13-Dezembro-2013]. 29, 37

[29] DataStax. DataStax OpsCenter. http://www.datastax.com/dev/blog/upgrading-an-existing-cluster-to-vnodes-2, 2014. [Online; acessado 13-Janeiro-2014]. vii, 30, 31

[30] DataStax. DataStax OpsCenter. http://www.datastax.com/what-we-offer/products-services/datastax-opscenter, 2014. [Online; acessado 13-Janeiro-2014]. 38

[31] C.J. Date. *Introdução a sistemas de bancos de dados*. Campus, 2004. 1, 15

[32] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, oct 2007. 23

[33] J. Ellis. Schema in Cassandra 1.1. http://www.datastax.com/dev/blog/schema-in-cassandra-1-1, 2013. [Online; acessado 3-Dezembro-2013]. 25, 26

[34] M. Fowler and P. J. Sadalage. *NoSQL distilled: a brief guide to the emerging world of polygot persistence*. Addison-Wesley Professional, 1 edition, 2012. 1, 17, 20, 21

[35] K. Gaj, T. El-Ghazawi, N. Alexandridis, J.R. Radzikowski, M. Taher, and F. Vroman. Effective utilization and reconfiguration of distributed hardware resources using job management systems. In *International Parallel and Distributed Processing Symposium.*, page 177, April 2003. 7

[36] A.S. Grimshaw and A. Natrajan. Legion: Lessons learned building a grid operating system. *Proceedings of the IEEE*, 93(3):589–603, March 2005. 7

[37] M. Gyssens, J. Paredaens, J. van den Bussche, and D. van Gucht. A graph-oriented object database model. *Knowledge and Data Engineering, IEEE Transactions on*, 6(4):572–586, 1994. 17

[38] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 336–341, 2011. 18, 19, 20, 21

[39] C. A. Heuser. *Projeto de Banco de Dados*. Sagra Luzzatto, 4 edition, 1998. vii, 14, 15, 16

[40] E. Hewitt. *Cassandra - The definitive Guide*. O'REILLY, 1st edition, 2010. 1, 23, 26, 27

[41] R. C. Huacarpuma. Modelo de dados para um pipeline de sequenciamento de alto desempenho transcritomico. Master's thesis, Universidade de Brasília, 2012. 33, 34, 39, 42

[42] N. Hurst. Visual Guide to NoSQL Systems. http://blog.nahurst.com/visual-guide-to-nosql-systems, 2013. [Online; acessado 3-Dezembro-2013]. vii, 22

[43] A. Kelly and D. McCreary. *Making Sense of NoSQL : A Guide for Managers and the Rest of Us by Ann Kelly and Dan McCreary*. Manning Publications Company, 2013. 17

[44] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010. 23

[45] D. C. Marinescu. *Cloud Computing: Theory and Practice*. Elsevier Science, Department of Electrical Engineering Computer Science. University of Central Florida, Orlando, FL 32816, USA, 1 edition, 2012. 7, 9

[46] P. Mell and T. Grance. The NIST definition of cloud computing. Technical report, National Institute of Standars and Technology, Information Technology Laboratory, July 2009. 1, 4, 10, 13

[47] J. Patel. Cassandra at NASA Meetup | DataStax HQ. http://www.planetcassandra.org/blog/post/cassandra-at-nasa-meetup-datastax-hq, 2013. [Online; acessado 3-Dezembro-2013]. 23

[48] J. Patel. Cassandra Data Modeling Best Practices, Part 1. http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part-1/, 2013. [Online; acessado 3-Dezembro-2013]. 23

[49] F. Prosdocimi. Bioinformática: Manual do usuário. *Biotecnologia Ciência e Desenvolvimento*, 29, nov 2002. 2

[50] J. W. Ross and G. Westerman. Preparing for utility computing: The role of it architecture and relationship management. *IBM Systems Journal*, 43(1):5–19, 2004. 9

[51] D. Sharma and P. Mittal. Grid computing an overview. *International Journal of Science, Engineering and Technology Research (IJSETR)*, 2(3):606–611, March 2013. 7, 8

[52] S. A. Simon, J. Zhai, R. S. Nandety, K. P. McCormick, J. Zeng, D. Mejia, and B. C. Meyers. Short-Read Sequencing Technologies for Transcriptional Analyses. *Annual Review of Plant Biology*, 60(1):305–333, jan 2009. 33

[53] F. R. C. Sousa, L. O. Moreira, and J. C. Machado. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *ERCEMAPI 2009, SBC*, pages 150–175, 2009. vii, 4, 5, 10, 11, 12

[54] A. Srivastava. NoSQL, NewSQL and MDM. `http://cdi-mdm.blogspot.com.br/2011/07/nosql-newsql-and-mdm.html`, 2013. [Online; acessado 06-Dezembro-2013]. vii, 19

[55] C. Strauch. *NoSQL Databases*. Stuttgart Media University, 2011. 17, 20

[56] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley Sons Inc., December 2002. 7

[57] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2008. viii, 6

[58] M. A. Vouk. Cloud computing x2014; issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31–40, 2008. 7, 8

[59] G. Wang and J. Tang. The nosql principles and basic application of cassandra model. In *Computer Science Service System (CSSS), 2012 International Conference on*, pages 1332–1335, 2012. 17

[60] C. S. Yeo, M. D. Assunção, J. Yu, A. Sulistio, S. Venugopal, M. Placek, and R. Buyya. Utility computing and global grids. Technical Report arXiv:cs/0605056v1, The University of Melbourne, April 2006. 8, 9